UNIVERSITY OF OSLO
Department of Informatics

Ars flectandi

Automated morphological analysis of Latin

Master's thesis

Arne Skjærholt

May 3, 2011

Ina, for not going insane when I did;

Jan Tore and Dag, for help with trees and forests;

Eman and Johan, everyone in the support group for frustrated students;

Thank you

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Latin is a dead language, but classics certainly aren't dead. Representing an almost unbroken tradition of research into Latin and Greek with a history whose length is measured in millennia, it is not surprising that it's a rather conservative field. Modern techniques and approaches have started to make their way into the field, however. New theories of grammar are applied to the classical languages, and treebanks of various parts of the Greek and Latin corpora are under construction. These treebanks are still small, however, and development is relatively slow. Issues such as funding and access to qualified labour are important factors in the development of these treebanks, but another important factor is the fact that the annotation process is entirely manual.

Computational linguistics on the other hand is a young field, not more than a few decades old. It is a synthesis[1] of ideas and approaches from a variety of different fields. Originally a sub-discipline of the research into artificial intelligence that started shortly after the second world war, it has grown into a separate field of inquiry, incorporating ideas from theoretical linguistics, machine learning and artificial intelligence research.

In the construction of a treebank of a classical language, the most time-consuming (and tedious) part of the annotation process is morphological annotation. As one might expect from a classical language, Latin has a rich and varied morphology, all aspects of which have to be recorded in the treebank. Tagging isn't glamorous work, but as long as the classical treebanks are so small that things like learning parsers are still in the future, we would like to develop tools that will accelerate the development of such corpora. This of course so that we can get to the good stuff faster.

This leads us to certain questions that must be answered regarding the automatic processing of Latin. What's a good approach to automatic analysis of Latin morphology, and what kind of performance can be considered good enough for an automatic analysis component to be useful in corpus construc-

[1] Or hodgepodge or mish-mash, depending on who you ask.

tion? And given that the data available for training statistical models is quite small, how severe will the effects of the small corpus and considerable size of the tagset be?

1.1 What?

We present here two different solutions to the two fundamental building blocks of an analysis solution for Latin: morpho-syntactic tagging and part-of-speech tagging. Both parts of the problem are solved as sequence classification problems, using both hidden Markov models and conditional random fields. Hidden Markov models are a well-known approach to the problem of sequence classification, their theory is simple and easily understood, and both parameter estimation and inference is simple and efficient. Conditional random fields are a more recent innovation in the field, first presented in 2001. The corresponding theory is more complicated than for hidden Markov models (but perhaps not as complicated as one might think), and they allow for models that exploit additional information that might be available for each word. Unfortunately parameter estimation is very expensive for conditional random fields.

Finally, we have evaluated an approach to make use of a large full-form dictionary of morphological analyses that is available. We do this by constraining the decoding process of the conditional random fields to only consider the tags licenced by the dictionary, rather than the several hundred tags in the full tagset.

1.2 Who?

The primary intended audience of this thesis is computational linguists, but in the hope that it may pique the interest of classicists of a modern bent as well. Being first and foremost for computational linguists, we assume a certain amount of prior knowledge. A certain insight into statistics and probability theory is mandatory for understanding statistical language models in general, but not much more than the basics should be required for the present work. Obviously a modicum of familiarity with the field of natural language processing is assumed, along with familiarity with the idea of sequence classification.

Some of the derivations, in particular those for the equations used in parameter estimation of CRFs (section 5.2), require some (very) basic multivariate calculus in the form of partial derivatives and the gradient. Also, the expressions involved are a bit hairy. Finally, a conversational knowledge of Latin and the concepts involved is helpful, but not strictly necessary to understand the meat of the matter.

1.3 Overview

Chapter 2 contains a brief overview of the history of Latin, its morphology and challenges, and the corpus used for the experiments.

Chapter 3 gives an introduction to graphical models, an important prerequisite for understanding conditional random fields as presented in chapter 5. Both directed and undirected models are presented.

Chapter 4 introduces hidden Markov models, both the general theory of the models and the peculiarities of the implementation used for the experiments.

Chapter 5 gives a quite thorough introduction to the theory of conditional random fields and the derivation of the most important equations involved.

Chapter 6 details the experiments performed, the results obtained, and presents a discussion of the results.

Chapter 7 explains the motivation for our experiments with constrained decoding of CRFs, the theoretical justification of the approach and the implementation, as well as an evaluation of the efficacy of the technique.

Chapter 8 sums up the results presented in the previous chapters, considers the question of whether HMMs or CRFs are to be preferred, and presents some possible avenues for future research.

Chapter 2

Language & corpus

In order that our discussions on statistical techniques for analysing Latin words don't take place in a complete vacuum, we start out with a brief discussion of Latin. First, where it comes from, a few important milestones in the history of the language, and a few examples before we take a closer look at Latin morphology, the corpus used to train the models, and some of the particular challenges of analysing Latin on the word level.

2.1 A brief history of Latin

The story of Latin begins in central Italy, some eight centuries BCE. The Latins are not very different from their neighbours, the Oscans and Umbrians, they even have similar languages. But through a combination of clever politicking and agressive imperialism the Latins and their language become dominant not only in Italy, but throughout Europe and the Mediterranean. The story ends sometime between 800 and 1600 CE, or might even continue today if we see the modern Romance languages as successors to Latin.

The earliest evidence of Latin as a written language dates to the sixth century BCE, and the earliest surviving examples of literature are the comedies of Plautus from around 200 BCE. The apex of Latin culture and literature however, was the classical period, a period of about two centuries from the first century BCE to the first century CE. This was the time of most Romans still known today such as Caesar and Augustus, Cicero and Virgil. The Latin of this period is a highly polished literary language, lush with stylistic sleight of hand and complicated syntactic structures.

A relatively simple example of Classical Latin is Caesar's *Commentarii de Bello Gallico* or *Bellum Gallicum* (usually abbreviated as *BG*), which is Caesar's account of how Gaul was conquered through a decade-long military campaign. Familiar to most who have studied Latin, it is written in a straightforward style, without many of the flourishes that complicate the reading of authors such as Cicero. As an example, the opening sentences of *Bellum Gal-*

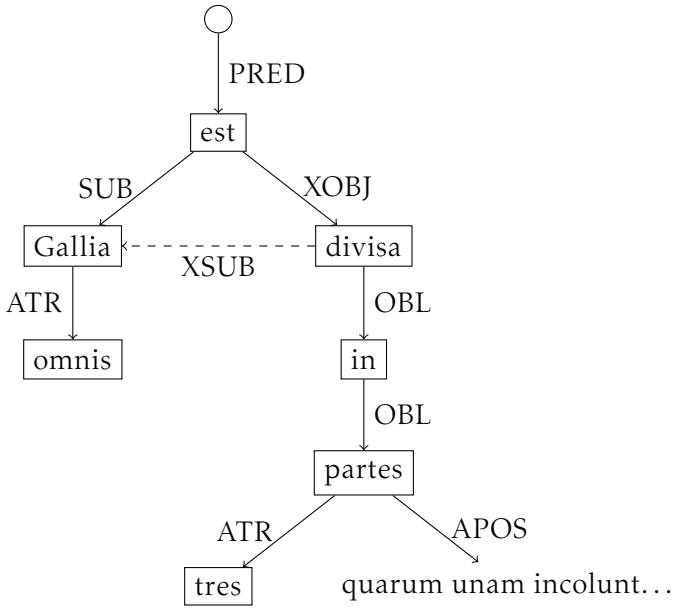

Figure 2.1: Dependencies in *BG* 1.1.1

licum are given with a translation below, with the syntactic relations of the opening words (corresponding to the first English sentence) in figure 2.1. The relations PRED and SUB identify the primary verb and its subject. ATR and APOS are used for attributes and appositions (ie. restrictive and non-restrictive noun qualifiers), respectively. Finally, the OBL relation is used for oblique (non-object) arguments, and XOBJ marks a complement whose open subject role is filled with the XSUB secondary dependency.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, institutis, legibus inter se differunt. Gallos ab Aquitanis Garumna flumen, a Belgis Matrona et Sequana dividit.

In all, Gaul is divided in three. Of these, the Belgians inhabit one, the Aquitans another, and those who are called Celts in their own language, or Gauls in our own, inhabit the third. All of them differ between each other in language, traditions and laws. The river Garonne separates Gauls from Aquitans, and the Seine and Marne from the Belgians.

The language of the people was quite different, even in Caesar's time, showing the first signs of the changes that would transform Latin into the Romance languages. This popular language is known as Vulgar Latin and

developed in parallel with the literary language; by the fifth century CE, the differences between literary Latin, which was still quite close to Classical Latin, and Vulgar Latin were so great that it is not entirely wrong to consider them different languages.

2.2 Latin morphology

Typologically, Latin is a highly inflecting language with a synthetic morphology, that is a high ratio of morphological features per morpheme. At the most abstract level, we can divide Latin morphology into two distinct classes: nominal and verbal inflection. The verbal inflection contains all the finite forms of the verb, while the nominal system handles nouns, adjectives, pronouns, and the non-finite forms of the verb. All Latin inflection is fundamentally concatenative, or based on attaching suffixes to an inflectional stem. In the most general case the construction of the stem is lexicalised, but most words have "siblings" that form their stems in the same way; likewise, which suffixes are used to form the various forms is a property of each individual word.

Nominal inflection, called declension, is the simpler of the two systems. Latin has three genders (masculine, feminine, neuter), two numbers (singular and plural) and six cases (nominative, vocative, accusative, genitive, dative, ablative). Nouns have inherent gender and are inflected in case and number, while adjectives and pronouns are inflected in all three dimensions. Adjectives are also inflected in degree (positive, comparative, superlative), and personal pronouns have an inherent number (first through third), just like in English. Adjectives agree in number, case and gender with the nominal they qualify; pronouns agree in number and gender with their antecedent and inflect in case according to their syntactic function.

There are five major classes, or declensions, of nominal inflection, each with their own independent sets of morphemes. The first three are used for both adjectives and nouns, while the fourth and fifth are minor classes only used for nouns. Pronominal inflection is irregular, and to a certain extent particular to each pronoun. A bird's eye view of the whole mess is quite uniform however. To generate a given case/number combination for a lemma, we first find the inflectional stem of the lemma and then append the morpheme corresponding to the case and number desired. Comparison of adjectives is marked with a separate morpheme sandwiched between the stem and the case-number morpheme.

Table 2.1 shows the full paradigm of the word *rosa* (en. *rose*), illustrating the declination of words according to the first declination. Hyphens separate the two morphemes of the various forms, and the lack of distinct number or case morphemes is apparent. An example adjective comparison is shown in table 2.2, which shows the feminine singular forms of *longus* (en. *long*)

	Sg.	Pl.
Nom.	ros-a	ros-ae
Voc.	ros-a	ros-ae
Acc.	ros-am	ros-as
Gen.	ros-ae	ros-arum
Dat.	ros-ae	ros-is
Abl.	ros-a	ros-is

Table 2.1: Noun paradigm: *rosa*

	Pos.	Comp.	Super.
Nom.	long-∅-a	long-ior-∅	long-issim-a
Voc.	long-∅-a	long-ior-∅	long-issim-a
Acc.	long-∅-am	long-ior-em	long-issim-am
Gen.	long-∅-ae	long-ior-is	long-issim-ae
Dat.	long-∅-ae	long-ior-i	long-issim-ae
Abl.	long-∅-a	long-ior-e	long-issim-a

Table 2.2: Adjective comparison: *longa*

in all three degrees. The positive and superlative use the case endings of the first declension, while the comparative uses the morphemes of the third or consonantic declension. Again, number and case is expressed through a single morpheme, while degree is expressed with a separate morpheme (the ∅ denotes a null morpheme).

While not as intimidating as Greek or Sanskrit, the Latin verbal system is quite complex, especially when compared to Germanic languages like English or Norwegian. The finite verb has 5 parameters: mood (indicative, subjunctive, imperative), tense (present, imperfect, future, perfect, pluperfect, future perfect), person (first to third), number (singular and plural), and voice (active and passive). Not all permutations of features are valid forms in Latin, but a regular verb has 98 possible forms out of the 216 possible.

Due to the complexity of the system, there is no neat classification of the Latin verbs. The basic pattern of the verbal morphology is the same as the nominal inflection: morphemes are appended to an inflectional stem. But unlike the nominals, a verb has three distinct stems. The present, imperfect and future tenses (collectively referred to as the *present system*) are formed using the present stem, while the perfect, pluperfect and future perfect tenses (the *perfect system*) are formed using the perfect stem. The final stem, the supine, isn't involved in the formation of finite forms, but is required to form various verbal nouns and participles.

The traditional classification of the verbs is into four conjugations, based

	Act.	Pass.
1st. sg.	ama-re-m	ama-re-r
2nd. sg.	ama-re-s	ama-re-ris
3rd. sg.	ama-re-t	ama-re-tur
1st. pl.	ama-re-mus	ama-re-mur
2nd. pl.	ama-re-tis	ama-re-mini
3rd. pl.	ama-re-nt	ama-re-rentur

Table 2.3: Verbal paradigm: *amare*

on a verb's behaviour in the present system. This classification is nice and orderly in the present system, but tends to break down for other forms (Ernout 1953, §171). In the first, second and fourth conjugations the system is mostly regular (with a certain number of exceptional verbs), but in the third conjugation the relationsip between between the three stems of a verb, present, perfect and supine, is almost entirely lexicalised. Perhaps not surprisingly, the morphemes involved in the verbal inflection are more overloaded than the morphemes of the nominal system. As an example we have the imperfect subjunctive forms of *amare* (en. *to love*), as shown in table 2.3. The morpheme *–re* marks the forms as imperfect subjunctive, while the other morphemes mark person, number and voice. This state of affairs is typical.

2.3 PROIEL

The Pragmatic Resources of Old Indo-European Languages (PROIEL) project aims to do a thorough comparative study of how pragmatic information is presented in old Indo-European languages. To this end, a parallel corpus of several classical Indo-European languages (Ancient Greek, Old Church Slavic, Classical Armenian, Gothic, and Latin) has been constructed. The main part of the corpus is translations[1] of the New Testament, but some some other texts are included as well that have no parallels in other languages.

The corpus is morphologically and syntactically annotated. The syntactic annotations are in the tradition of dependency grammar, extended with secondary dependencies similar to structure sharing in LFG and HPSG (Haug et al. 2009). The guidelines for annotation[2] are derived from the annotation guidelines of the Perseus Latin Treebank[3], themselves derived from the Prague Czech treebank annotation guidelines[4].

[1] Or original, in the case of Ancient Greek

[2] http://folk.uio.no/daghaug/syntactic_guidelines.pdf

[3] http://nlp.perseus.tufts.edu/syntax/treebank/latin.html

[4] http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/a-layer/html/index.html

Corpus	Sentences	Tokens	Avg. tok/sen
BG	1,296	25,167	19.4
Vulgata	12,435	113,079	9.1
Peregrinatio	921	18,351	19.9
Total	14,652	156,597	10.7

Table 2.4: Corpus sizes

The Latin part of the corpus is made up of three texts: the *Vulgata* translation of the New Testament, Caesar's *Bellum Gallicum*, and *Peregrinatio Aetheriae*, a fifth century Vulgar Latin account of a pilgrimage to the Holy Land. Of these, the *Vulgata* is by far the largest, while *BG* and *Peregrinatio* are significantly smaller. See table 2.4 for detailed statistics. As mentioned above, Vulgar Latin is very different from the Classical Latin of Caesar and the literary style of the *Vulgata*, and for this reason the *Peregrinatio* corpus is not used for any of our experiments.

2.3.1 Tagsets

There are two tagsets in the corpus, a part-of-speech (PoS) tagset and a morpho-syntactic descriptor (MSD) tagset, both with the same basic structure: fixed width strings where each character encodes a field. The PoS tagset has two fields, a major tag that places the word in one of eleven rough classes (the 10 parts-of-speech of traditional grammar with an additional class for foreign words), and a minor tag that further subdivides some of the classes. See table 2.14 for a list of all the PoS tags used in the Latin part of PROIEL and their meanings.

The MSD tagset is ten characters wide. The eight first fields encode the various morphological parameters of Latin in the order person, number, tense, mood, voice, gender, case, and degree. The ninth field is unused in the Latin part of the corpus[5], and the final field encodes whether the form is inflecting or not. Tables 2.5 to 2.13 give the values the various fields can have. Null fields in both tagsets are represented using –. The ambiguous gender tags o, p, q, and r are used to avoid making arbitrary choices regarding congruence in the morphological annotation.

2.4 What makes Latin hard

There are two main challenges in the analysis of Latin morphology: syncretism and clitics. Syncretism is the term used in linguistics to describe

[5]It encodes strong/weak inflection in the Gothic and Old Church Slavonic parts of the corpus

Person	
1	1st person
2	2nd person
3	3rd person

Table 2.5: Person

Number	
s	singular
p	plural

Table 2.6: Number

Tense	
p	present
i	imperfect
f	future
r	perfect
l	pluperfect
t	future perfect

Table 2.7: Tense

Mood	
i	indicative
s	subjunctive
m	imperative
n	infinitive
p	participle
d	gerund
g	gerundive
u	supine

Table 2.8: Mood

Voice	
a	active
p	passive

Table 2.9: Voice

Gender	
m	masculine
f	feminine
n	neuter
o	m/n
p	m/f
q	m/f/n
r	f/n

Table 2.10: Gender

Case	
n	nominative
v	vocative
a	accusative
g	genitive
d	dative
b	ablative

Table 2.11: Case

Degree	
p	positive
c	comparative
s	superlative

Table 2.12: Degree

Inflection	
i	Inflected
n	Non-inflected

Table 2.13: Inflection

Tag	Meaning	Tag	Meaning
A–	adjective	Pc	reciprocal pronoun
C–	conjunction	Pd	demonstrative pronoun
Df	adverb	Pi	interrogative pronoun
Dq	relative adverb	Pk	personal reflexive pronoun
Du	interrogative adverb	Pp	personal pronoun
F–	foreign word	Pr	relative pronoun
G–	subjunction	Ps	possessive pronoun
I–	interjection	Pt	possessive reflexive pronoun
Ma	cardinal numeral	Px	indefinite pronoun
Mo	ordinal numeral	R–	preposition
Nb	common noun	V–	verb
Ne	proper noun		

Table 2.14: PROIEL PoS tags

	Sg.	Pl.
Nom.	fruct-us	fruct-us
Voc.	fruct-us	fruct-us
Acc.	fruct-um	fruct-us
Gen.	fruct-us	fruct-uum
Dat.	fruct-ui	fruct-ibus
Abl.	fruct-u	fruct-ibus

Table 2.15: Syncretic paradigm: *fructus*

the case where several morphological forms of a word are identical. This phenomenon is commonly found in fusional languages like Latin, and it is widespread in Latin. In the nominal system, words without ambiguous forms in their paradigm are rare birds indeed. The most pronounced example of this is the fourth declension paradigm *fructus* (en. *fruit*) shown in table 2.15, where half the possible forms are identic. The verb paradigms are also syncretic, but not quite to the extent of the nominal system. This is an obvious (and classic) problem for tagging classifiers.

Clitics are unaccented words that share some features of morphemes, but are syntactically independent. The abbreviated forms of the auxiliary verbs in English (*'ll*, *'ve*, etc.) are examples of this; they participate independently in syntax, but have to be attached to another word because they aren't phonologically independent. Latin has five such clitics: the conjunction *que*, the disjunction *ve*, the interrogative *ne*, the emphatic *met*, and the preposition *cum* which can also be used as a clitic. They don't pose problems for tagging

proper, but rather to tokenisation; they aren't written separately in running text, nor are there any distinguishing marks like the apostrophe in English to guide tokenisation.

Chapter 3

Graphical models

Graphical models are a way of representing collections of random variables and the relationships between them. Originally studied in the field of statistical mechanics (the Ising model of ferromagnetism is a notable example), they have turned out to be a useful generalisation of many statistical models used in machine learning and related disciplines.

Before moving on to the actual theory, a few matters of notation. As is normal, random variables are set in capital letters (X, Y, Z) and particular instantiations of these in lower case (x, y, z). For sets of variables, we follow the notation of Pearl (1988): variable sets are set in boldface capitals ($\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$) and assignments of values to each of the variables in a set (configurations) in the corresponding minuscule boldface letter ($\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$).

3.1 Graphs

A graph is defined as a pair $G = (V, E)$, where V is the set of *nodes* (or vertices) in the graph and E is the set of *edges*, which specify which nodes are connected. Each edge is a pair $(u, v) \in V^2$, but the interpretation of an edge depends on whether the graph is *directed* or *undirected*. In an undirected graph, the edge (u, v) means that v is reachable from u and vice versa. In a directed graph on the other hand, it only means that v is reachable from u; u is reachable from v only if the edge (v, u) is also in E. (Diestel 1991, 2–4, 23)

A *path* through a graph is a sequence of nodes $v_1, \ldots, v_n$ such that each node v_i after the first is reachable from the previous one and all the nodes are distinct, except v_1 and v_n which may be the same node. A path that starts and ends in the same node is called a *cycle*. The special case of a directed graph with no cycles is called a *directed acyclic graph* (DAG), and they are common enough that they have their own terminology. In particular, the lack of cycles makes it meaningful to talk about the *parents* of a node: the set of nodes such that v is reachable from them, we denote this set $v_\pi = \{u | (u, v) \in E\}$. An undirected graph with no cycles is called a *tree*. (Diestel 1991, 6–7, 12)

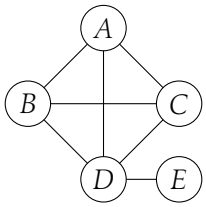

Figure 3.1: A simple undirected graph

A *clique* is a completely connected subgraph, that is, a subset C of the nodes of a graph such that every node in the clique is adjacent to every other node. A *maximal* clique is a clique that cannot be made larger by adding another node, or equivalently a clique which isn't a subset of any other clique in the graph (Wallach 2002, 25). For example, in figure 3.1, $\{A, B, C\}$ is a clique, but not a maximal clique since it is contained in the larger clique $\{A, B, C, D\}$. $\{C, D, E\}$ is not a clique because C and E are not adjacent. We will use the notation $C(G)$ for the set of maximal cliques in G and V_c for the nodes of a clique c.

3.2 Graphical models

Graphical models are a convenient tool for representing collections of random variables and the dependencies between them. We do this by combining a set of random variables $\boldsymbol{X}$ and a graph $G = (V, E)$ such that there is a one-to-one mapping between $\boldsymbol{X}$ and V, and the edges of the graph correspond to the conditional dependencies between the variables. This mapping blurs the line between the two, leading to an ambiguity we will happily exploit to make notation simpler. For example, we will write $C(\boldsymbol{X})$ to denote the cliques in the graph backing the model of the variable set $\boldsymbol{X}$, or use nodes in the place of corresponding variable in probabilities.

An important concept when discussing graphical models is that of *conditional independence*. Two variable sets $\boldsymbol{X}$ and $\boldsymbol{Y}$ are conditionally independent, given a set $\boldsymbol{Z}$ iff $p(\boldsymbol{X}|\boldsymbol{Y}, \boldsymbol{Z}) = p(\boldsymbol{X}|\boldsymbol{Z})$ (Pearl 1988). This is an extension of the idea of marginal (or unconditional) independence, where two variables X and Y are marginally independent if and only if $p(X|Y) = p(X)$[1].

3.2.1 Directed models

In a directed graphical model, we use a DAG to represent the dependencies between the variables in $\boldsymbol{X}$. A simple example that can be represented as such

[1] Usually this is formulated as $p(X, Y) = p(X)p(Y)$, but for the notion of conditional independence, the equivalent $p(X|Y) = p(X)$ is more illustrative.

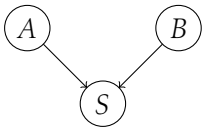

Figure 3.2: A simple directed model

a model is the sum of two dice, the model in figure 3.2: The sum S depends on the outcomes of the dice A and B. As the figure shows, dependencies flow in the direction of the edges in the graph.

In a directed model, for any variable X the parents X_π of that node are the minimal set of nodes required to make X conditionally independent of the remaining nodes in G (Wallach 2002). The upshot of this is that computing the probability of any given configuration is a quite straightforward affair. For each variable, we compute the probability of its value, given the values of its parents:

$$p(\boldsymbol{X}) \triangleq \prod_{X \in \boldsymbol{X}} p(X|X_\pi) \tag{3.1}$$

3.2.2 Undirected models

As the name implies, undirected models use undirected graphs to represent the dependencies between variables. In such a model, an edge between two nodes means that two adjacent nodes influence the other, which in turn means that undirected models can represent different classes of dependencies than directed models.

Similar to the directed case, given its neighbouring nodes, any variable X is conditionally independent of all the remaining nodes in the model. Taking inspiration from equation (3.1) it would be tempting to define distributions for each X given the variable's neighbours δX: $p(X|\delta X)$. However, since X influences its neighbours as well, making sure that the graph structure and the actual distributions are consistent is a non-trivial exercise in this scheme (Pearl 1988, 105). A better approach is required.

The solution to the problem is to define a *Gibbs' potential* Ψ over the graph. This potential is the product of individual potential functions Ψ_c, each defined over the variables of the maximal cliques $c \in C(G)$ of the graph.

$$\Psi(\boldsymbol{X}) \triangleq \prod_{c \in C(\boldsymbol{X})} \Psi_c(\boldsymbol{X}_c) \tag{3.2}$$

The only constraint on the potential functions is that their values be strictly positive. But since the potential functions can have any value greater than zero, Ψ will not in general be a valid probability, so the potential has to be normalised to yield a probability. This normalisation factor Z is the sum of

the potentials of all $\boldsymbol{x} \in \Omega(\boldsymbol{X})$, where $\Omega(\boldsymbol{X})$ is the set of all possible assignments of values to the variables of $\boldsymbol{X}$:

$$Z(\boldsymbol{X}) \triangleq \sum_{\boldsymbol{x} \in \Omega(\boldsymbol{X})} \Psi(\boldsymbol{x}) \tag{3.3}$$

This gives us the complete expression for the probability of an undirected graphical model:

$$p(\boldsymbol{X}) \triangleq \frac{1}{Z(\boldsymbol{X})} \Psi(\boldsymbol{X}) \tag{3.4}$$

Chapter 4

Hidden Markov models

Hidden Markov models (HMMs) are one of the simplest language modeling techniques, and commonly used for sequence classification problems. In this model, the words of a sentence are seen as the outputs from a series of random states. The word emitted from any given state depends only on the current state, and likewise the probability of the next state depends only on the current state. The problem is then to determine the most probable sequence of states for a given sequence of observed emissions.

Before we formalise HMMs, we'll get some notation out of the way. $V = \{w_1, \ldots, w_n\}$ denotes a finite alphabet; $W = w_1 \cdots w_T$ denotes a sequence of length T, $\mathcal{Q} = \{q_1, \ldots, q_n\}$ is a finite set of states, and $Q = q_1 \cdots q_T$ a sequence of states. $\mathbb{P} = [0, 1]$ is the set of valid probabilities.

4.1 Definition

We can then define a HMM as a four-tuple $H = (\mathcal{Q} \cup \{q_s, q_e\}, V, t, e)$, where the start and end states $q_s, q_e \notin \mathcal{Q}$. t is the transition function $t : \mathcal{Q} \cup \{q_s\} \times \mathcal{Q} \cup \{q_e\} \to \mathbb{P}$, where $t(q', q)$ gives $p(q|q')$, the probability of moving from the state q' to q, and e the emission function $e : \mathcal{Q} \times V \to \mathbb{P}$ for $p(w_i|q_i)$, the probability that the state q emits the word w (Brants 1999, 16). Finally, the transition and emission functions have to sum to 1 in the appropriate way to make sure they give proper probabilities (Jurafsky and Martin 2008, 211):

$$\sum_{q' \in \mathcal{Q} \cup \{q_e\}} t(q, q') = 1 \quad \text{For all } q \in \mathcal{Q} \cup \{q_s\}$$

$$\sum_{w \in V} e(q, w) = 1 \quad \text{For all } q \in \mathcal{Q}$$

Using q_0 as a synonym of q_s for simplicity's sake, the probability of a label sequence-output pair is:

$$p(Q, W) = p(q_e|q_T) \prod_{t=1}^{T} p(q_t|q_{t-1}) p(w_t|q_t) \tag{4.1}$$

We then need to estimate the various probabilities of our model, which is usually done by simply counting occurrences in a training corpus. Using $\hat{p}$ to distinguish the estimate from the true probability, the transition probabilities are estimated by $\hat{p}(q|q') = c(q', q)/c(q')$ and the emission probabilities by $\hat{p}(w|q) = c(w, q)/c(q)$. The function c counts the number of times various configurations occur in the corpus: $c(q', q)$ the number of times the label q' is followed by q, $c(q)$ the number of times the label q appears, and $c(w, q)$ how many times word w has the label q. These estimates are quite good, especially with a decent sized corpus, as well as very simple to compute. They are Maximum Likelihood Estimates (MLEs) obtained by seeing the corpus as being produced by a multinomial distribution; the full derivation of the estimates is given in appendix A.

An important question in statistical language modeling is how to handle unknown words. In the model above, an input string that contained a word w_u not encountered in training would have probability 0 for all possible label sequences, since $p(w_u|q) = 0$ for any state q. Usually, this problem is handled by creating a special vocabulary item to handle such words and use a discounting technique such as add-1 smoothing or Good-Turing discounting to assign emission probabilities to unknown words for all the states. The similar problem of some state transitions being unobserved in training is handled in an analog manner. (Jurafsky and Martin 2008, 131–137)

An HMM where the transition probabilities depend on the previous and current states is called a first-order, or bigram, HMM. Higher order HMMs use the n previous states to compute the transition probabilities. A second order HMM has $p(q_i|q_{i-1}, q_{t-2})$, and so on. These models are interesting for tagging as they can capture longer range dependencies than the basic first-order HMM. Conveniently, an n^{th}-order HMM with k states can be represented as a first-order HMM with k^n states, where the different states encode different histories. The higher-order notation is simpler to work with, however. (Brants 1999, 18)

4.1.1 As graphical model

HMMs can also be defined as a directed graphical model (Wallach 2002, 11). For each word i in the sequence to be tagged, there are two nodes q_i and w_i. The node q_i is the parent of w_i, and $q_i, i < T$ is the parent of q_{i+1}. The start and end states are represented as the nodes q_s and q_e, where q_s is the parent of q_1 and q_T the parent of q_e. Figure 4.1 show the graph for a four-word

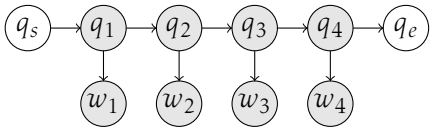

Figure 4.1: HMM as graphical model

sentence. The variables corresponding to the grey nodes are the variables whose distributions are modeled, while the distributions of the unshaded nodes aren't.

We then apply equation (3.1) to this graph and get the probability of a given tag sequence (again with q_0 as a synonym of q_s):

$$p(Q, W) = p(q_e|q_T) \prod_{i=1}^{T} p(q_i|q_{i-1}) p(w_i|q_i) \tag{4.2}$$

which is exactly the same as (4.1).

4.2 Decoding

Now that we have the theoretical machinery in place, all that's missing is a concrete means to find the most probable tag sequence for a given sequence of words, since simply enumerating all possible tag sequences (of which there are $|\mathcal{Q}|^T$) is infeasible for all but the very shortest strings. The solution is to use a dynamic programming approach called *Viterbi's algorithm*.

The trick is the realisation that if we incrementally compute probabilities for labelings of subsequences, we can compute the probability of a longer sequence using the shorter ones. The probability of the first word having a given tag is simply $p_{q,1} = t(q_s, q)e(q, w_1)$, for any q. The probability of any tag for the second word is then $p_{q,2} = \max_{q' \in \mathcal{Q}} p_{q',1} t(q', q)e(q, w_2)$, and so on for longer and longer subsequences until we have the probability of the entire sequence. More formally, we define the probability $\delta_t(q)$ of token t in the input having the label q as a recursive function: (Brants 1999, 18–20; Jurafsky and Martin 2008, 218–220)

$$\delta_t(q) = \begin{cases} t(q_s, q)e(q, w_1) & t = 1 \\ \max_{q' \in \mathcal{Q}} \delta_{t-1}(q')t(q', q)e(q, w_t) & t > 1 \end{cases} \tag{4.3}$$

and $\max_{Q \in \mathcal{Q}^T} p(Q, W) = \max_{q' \in \mathcal{Q}} \delta_T(q')t(q', q_e)$ gives the optimal sequence of states. Finally, we need to keep track of which states q_t gave the best probabilities:

$$q_t = \begin{cases} \operatorname{argmax}_{q \in \mathcal{Q}} \delta_T(q)t(q, q_e) & t = T \\ \operatorname{argmax}_{q \in \mathcal{Q}} \delta_t(q)t(q, q_{t+1}) & t < T \end{cases} \tag{4.4}$$

In implementation terms, this means that for all positions in the string, we need to store the highest possible probability of that tag being output, and coming from which previous state gives that probability. The natural way to do this is as a $|\mathcal{Q}| \times T$ matrix, usually referred to as the trellis. Each entry in the trellis is a pair (q, p) which stores the highest probability p of being in a particular state at a particular time, and the previous state q that gives that probability. Since the algorithm requires looping over all possible previous states for each possible current state at each point in the input, Viterbi's algorithm has an $O(|\mathcal{Q}|^2 T)$ asymptotic run-time. Figure 4.2 gives pseudo-code for the algorithm, assuming states are consecutive integers, so that they can be used to index the matrix.

Input: Input sequence $w_1 \cdots w_T$
Data: trellis, a $|\mathcal{Q}| \times T$ matrix
`# Initialisation`
for $q \in \mathcal{Q}$ **do**
  $\mathsf{trellis}_{q,1} = (q_s, e(q, w_1) * t(q_s, q))$
end
for $2 \leq i \leq T$ **do**
  for $q \in \mathcal{Q}$ **do**
    $p = \max_{q' \in \mathcal{Q}} p(\mathsf{trellis}_{q',i-1}) * t(q', q) * e(q, w_i)$
    $q' = \operatorname{argmax}_{q' \in \mathcal{Q}} p(\mathsf{trellis}_{q',i-1}) * t(q', q)$
    $\mathsf{trellis}_{q,i} = (q', p)$
  end
end
$q' = \operatorname{argmax}_{q' \in \mathcal{Q}} p(\mathsf{trellis}_{q',T}) * t(q', q_e)$
return *The most likely path through the trellis by following the back pointers, starting with* q'

Figure 4.2: Viterbi's algorithm for HMMs

4.3 Trigrams'n'Tags

Trigrams'n'Tags (TnT) is a sophisticated trigram (second-order) HMM tagger, authored by Thorsten Brants, and described in depth in Brants (2000). TnT is a relatively straightforward affair, but one particular part is of interest: its handling of unknown words. Instead of simply creating an unknown word vocabulary item and assigning it a probability based on discounting, TnT estimates the probabilities based on the suffixes of the word, a highly competitive strategy for suffix-inflecting languages like Latin.

TnT's implementation of this strategy uses low-frequency (occurring not

more than 10 times) words to build a suffix-trie[1] which is then used to compute the probabilities for each suffix. The probability of a tag q given the n last letters of a word, $s_n \cdots s_1$, is computed from the MLE for the suffix by linear abstraction (Samuelsson 1996) of the probabilities of the shorter suffixes according to the recursive function:

$$p(q|s_n \cdots s_1) = \frac{\hat{p}(q|s_n \cdots s_1) + \theta p(q|s_{n-1} \cdots s_1)}{\theta + 1} \tag{4.5}$$

using $\hat{p}(q|s_n \cdots s_1) = {}^{c(q,s_n \cdots s_1)}\!/_{c(s_n \cdots s_1)}$, the number of times the suffix occurs with the label divided by the number of times the suffix appears, to estimate the probability of a tag given a suffix, and $p(q|s_0 \cdots s_1) = p(q) = \hat{p}(q)$, the unigram tag MLE as the base case. The weight θ is given by:

$$\theta = \frac{1}{s-1} \sum_{i=1}^{s} (\hat{p}(q_i) - \bar{p}) \tag{4.6}$$

$$\bar{p} = \frac{1}{s} \sum_{i=1}^{s} \hat{p}(q_i) \tag{4.7}$$

where s is the size of the tagset. TnT uses a maximum suffix length of 10, and the emission probability of an unknown word is then the estimated probability of the longest suffix of the word which was observed in the corpus.

[1] A trie is a data-structure similar to a B-tree which provides efficient storage of strings that have similar prefixes or suffixes

Chapter 5

Conditional random fields

Conditional random fields (CRFs) are a more recent innovation in the field of statistical language modeling, and like HMMs they are most commonly used for sequence labeling tasks. A CRF is a discriminative model, as opposed to a HMM which is generative, which means that the joint distribution $p(W, Q)$ of observation and label sequence is not explicitly modeled, but rather the distribution $p(Q|W)$. This makes it possible for the discriminative model to incorporate several kinds of features on each observation, something that is not easily done in a generative model. In a generative model one would either have to make unwarranted independence assumptions, or model increasingly complicated (and sparse) joint distributions. On the other hand, parameter estimation is correspondingly more expensive, which is a practical cost that has to be justified.

5.1 Definition

CRFs are a class of undirected graphical model as presented in 3.2, but with additional constraints on the structure of the underlying graph to make inference and parameter estimation more tractable. They were first presented in Lafferty, McCallum and Pereira (2001), with the motivation that they avoid the label bias problem of MaxEnt Markov Models, where states with low entropy in the transition distributions "take little notice of observations". They define a CRF as follows:

Definition 1 *Let $G = (V, E)$ be a graph such that $\boldsymbol{Y} = (\boldsymbol{Y}_v)_{v \in V}$, so that $\boldsymbol{Y}$ is indexed by the vertices of G. Then $(\boldsymbol{X}, \boldsymbol{Y})$ is a* conditional random field *in case, when conditioned on $\boldsymbol{X}$, the random variables $\boldsymbol{Y}_v$ obey the Markov property with respect to the graph: $p(\boldsymbol{Y}_v|\boldsymbol{X}, \boldsymbol{Y}_w, w \neq v) = p(\boldsymbol{Y}_v|\boldsymbol{X}, \boldsymbol{Y}_w, w \sim v)$, where $w \sim v$ means that w and v are neighbors in G.*

While seemingly a forbidding definition, it turns out to be reasonably simple when you get down to it. The first sentence simply means that a CRF

is a kind of graphical model. The second sentence then states the Markov condition for CRFs: For any two output nodes v and w that are not neighbours in the graph, there exists a single neighbour w' of v such that v and w are conditionally independent given w' and $\boldsymbol{X}$. This simply means that the nodes of $\boldsymbol{Y}$ have to form a tree. A simpler definition is:

Definition 2 *A CRF is an undirected graphical model, such that when conditioned on $\boldsymbol{X}$ the following property holds for all nodes in $\boldsymbol{Y}$: for a pair of variables (Y_v, Y_w) there exists a single neighbour $Y_{w'}$ of Y_v such that $p(Y_v|\boldsymbol{X}, Y_w) = p(Y_v|\boldsymbol{X}, Y_{w'})$; that is, the nodes of $\boldsymbol{Y}$ must form a tree.*

Since we require $\boldsymbol{Y}$ to be tree-structured, the largest possible clique in the output graph is a pair of adjacent nodes. According to the Clifford-Hammersley theorem (Clifford 1990, 22) the probability $p(\boldsymbol{Y}|\boldsymbol{X})$ is then on the form

$$p(\boldsymbol{Y}|\boldsymbol{X}) \propto \exp\left(\sum_{e\in E,k} \lambda_k f_k(e, \boldsymbol{Y}|_e, \boldsymbol{X}) + \sum_{v\in V,k} \mu_k g_k(v, \boldsymbol{Y}_v, \boldsymbol{X})\right) \tag{5.1}$$

where $\boldsymbol{Y}|_e$ is the two endpoints of the edge e, and the f_k and g_k are fixed feature functions. In HMM terms, the edge features f_k can be seen as modulating the transition probabilities between neighbouring states, and the vertex features g_k as emission functions. The feature functions can be any strictly positive, real-valued function, but in most applications they are binary; if some configuration observed in the training data is seen in the input the function returns 1, and 0 in any other case.

The simplest possible tree is a first-order chain as shown in figure 5.1, where each node $\boldsymbol{Y}_i$ is connected to the previous node. As in 4.1.1, the shaded nodes are the ones whose distributions are modeled. In this case equation (5.1) can be simplified to:

$$\begin{aligned} p(\boldsymbol{Y}|\boldsymbol{X}) &\propto \exp\left(\sum_{i=1}^{T}\sum_{k} \lambda_k f_k(\boldsymbol{Y}_{i-1}, \boldsymbol{Y}_i, \boldsymbol{X}, i) + \sum_{k} \mu_k g_k(\boldsymbol{Y}_i, \boldsymbol{X}, i)\right) \\ &= \exp\left(\sum_{i=1}^{T}\sum_{k} \theta_k f_k(\boldsymbol{Y}_{i-1}, \boldsymbol{Y}_i, \boldsymbol{X}, i)\right) \\ &= \exp\left(\sum_{k} \theta_k F_k(\boldsymbol{Y}, \boldsymbol{X})\right) \qquad F_k(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{i=1}^{T} f_k(\boldsymbol{Y}_{i-1}, \boldsymbol{Y}_i, \boldsymbol{X}, i) \end{aligned} \tag{5.2}$$

with the additional simplification of writing the vertex features on the same form as the edge features. The node vertex features will then obviously simply ignore the previous node argument, and this simplification is purely notational to make the expressions shorter. Additionally, chain-structured CRFs are augmented with start and end states, like HMMs. The rest of this

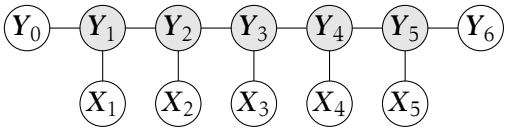

Figure 5.1: A chain-structured CRF

chapter will deal exclusively with such CRFs, augmented with a node $\boldsymbol{Y}_0$ whose label is always the start and $\boldsymbol{Y}_{n+1}$ whose label is stop. We assume that the labels start and stop are not already in $\mathcal{Q}$, and they serve the same purpose as the special states q_s and q_e for HMMs.

To get probabilities from the previous expressions, the values have to be normalised by the input-dependent normalisation factor $Z(\boldsymbol{X})$. As can be expected from (3.3), the value of this factor is obtained by summing the potentials of all possible label assignments $\boldsymbol{y}$ for the input:

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{Q}^T} \exp\left(\sum_k \theta_k F_k(\boldsymbol{x}, \boldsymbol{y}) \right) \tag{5.3}$$

5.2 Parameter estimation

Now that we know how to compute the probability of a tag sequence for a given input, the next step is to estimate the values of the parameters θ_k of the model. For CRFs we use maximum-likelihood estimation, which can be shown to give the distribution with maximum entropy (Wallach 2004).

Likelihood is a measure of the probability of a set of parameter values for a model with respect to some training data. Given a set of training data $\mathcal{D} = \{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\}$, and a fixed set of feature functions f_k, the likelihood of a parameter set is the product of the probabilities a model with those parameters assigns to the training examples: $L(\theta) = \prod_k p_\theta(\boldsymbol{y}^{(k)} | \boldsymbol{x}^{(k)})$, where p_θ is the probability according to the model with parameters θ.

Being a very large product, the likelihood is rather unwieldy to work with. This problem, however, is easily remedied; applying the logarithm will convert the product into a sum of logarithms. Additionally, the likelihood is of little interest to us. We simply want to find the parameters that yield the largest likelihood, and since the logarithm is strictly increasing, finding the parameters that maximise the log-likelihood $\mathcal{L}(\theta) = \log L(\theta)$ will yield the same parameters as studying L directly.

The log-likelihood of a CRF parameter set is then:

$$\mathcal{L}(\theta) = \sum_k \sum_j \theta_j F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}) - \sum_k \log Z(\boldsymbol{x}^{(k)})$$

a continuous and concave function (Lafferty, McCallum and Pereira 2001). Since the function is concave, finding the maximum is simply a matter of solving the equation $\nabla\mathcal{L} = 0$, which is equivalent to solving $\partial/\partial\theta_j\mathcal{L} = 0$ for all the features f_j. Before deriving $\partial/\partial\theta_j\mathcal{L}$, we find $\partial/\partial\theta_j Z$:

$$\begin{aligned}\frac{\partial}{\partial\theta_j}Z(\boldsymbol{x}^{(k)}) &= \sum_{k'}\frac{\partial}{\partial\theta_j}\exp\left(\sum_j \theta_j F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k')})\right) \\ &= \sum_{k'} F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k')})\exp\left(\sum_j \theta_j F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k')})\right)\end{aligned}$$

Using this, we find the derivative of the log-likelihood function

$$\begin{aligned}\frac{\partial}{\partial\theta_j}\mathcal{L}(\theta) &= \frac{\partial}{\partial\theta_j}\sum_k\sum_j \theta_j F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}) - \frac{\partial}{\partial\theta_j}\sum_k \log Z(\boldsymbol{x}^{(k)}) \\ &= \sum_k F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}) - \sum_k \frac{\partial/\partial\theta_j Z(\boldsymbol{x}^{(k)})}{Z(\boldsymbol{x}^{(k)})} \\ &= \sum_k F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}) - \sum_k\sum_{k'} \frac{F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k')})\exp\left(\sum_j \theta_j F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k')})\right)}{Z(\boldsymbol{x}^{(k)}} \\ &= \sum_k F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}) - \sum_k\sum_{k'} F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k')}) p_\theta(\boldsymbol{y}^{(k')}|\boldsymbol{x}^{(k)}) \\ &= \sum_k F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}) - \sum_k E_{p(\boldsymbol{Y}|\boldsymbol{x}^{(k)})}[F_j(\boldsymbol{x}^{(k)}, \boldsymbol{Y})]\end{aligned}$$

where $E_p[\cdot]$ is the expected value of $\cdot$ with respect to the distribution p.

Unfortunately, $\nabla\mathcal{L} = 0$ is not analytically solvable in the general case. Thus, we need to use numerical approximation techniques to find the best parameter values; there are a variety of algorithms available, each with different strengths and weaknesses, but the most popular seem to be quasi-Newton methods such as L-BFGS or hillclimbing methods such as variations on gradient ascent.

5.3 Decoding

Despite the theory being more involved than for HMMs, chain-structured CRFs can be decoded using essentially the same Viterbi algorithm as presented in figure 4.2. The principle remains the same: moving left to right in the input, we find the best tag sequences for subsequences of increasing length, until we've found the best sequence for the whole input.

We define $M_i(\boldsymbol{x})$, a $|\mathcal{Q}| \times |\mathcal{Q}|$ matrix, for each word in the sentence. The individual elements of the M_i are given by $M_i(q', q|\boldsymbol{x}) = \exp(\sum_k \theta_k f_k(q', q, \boldsymbol{x}, i))$, the potential of the assignment $\boldsymbol{Y}_{i-1} = q', \boldsymbol{Y}_i = q$. We can then decode the

CRF in the normal Viterbi fashion, multiplying potentials as we go, which gives us the algorithm in figure 5.2. The normalisation factor $Z_\theta(\boldsymbol{x})$ is the $\mathsf{start}, \mathsf{stop}$ entry of the matrix product $\prod_{i=1}^{T} M_t(\boldsymbol{x})$. (Lafferty, McCallum and Pereira 2001)

Input: Input sequence $w_1 \cdots w_T$
Data: trellis, a $|\mathcal{Q}| \times T$ matrix
`# Initialisation`
for $q \in \mathcal{Q}$ **do**
  $\mathsf{trellis}_{q,1} = (q_s, M_1(\mathsf{start}, q|\boldsymbol{x}))$
end
for $2 \leq i \leq T$ **do**
  for $q \in \mathcal{Q}$ **do**
    $p = \max_{q' \in \mathcal{Q}} p(\mathsf{trellis}_{q',i-1}) * M_i(q', q|\boldsymbol{x})$
    $q' = \operatorname{argmax}_{q' \in \mathcal{Q}} p(\mathsf{trellis}_{q',i-1}) * M_i(q', q|\boldsymbol{x})$
    $\mathsf{trellis}_{q,i} = (q', p)$
  end
end
$q' = \operatorname{argmax}_{q' \in \mathcal{Q}} p(\mathsf{trellis}_{q',T}) * M_{T+1}(q', \mathsf{stop}|\boldsymbol{x})$
return *The most likely path through the trellis by following the back pointers, starting with* q'

Figure 5.2: Viterbi's algorithm for CRFs

In addition to the decoding algorithm, there's something we glossed over in the previous section. The log-likelihood function that we optimise requires us to compute $E_{p(\boldsymbol{Y}|\boldsymbol{x}^{(k)})}[F_j(\boldsymbol{x}^{(k)}, \boldsymbol{Y})]$ for all the training examples. Doing this even once in the naïve way is intractable for the same reason the simple approach to decoding doesn't work: there's an exponential number of label sequences. Additionally, we have to do this several times, once for each iteration of the numerical approximation algorithm used. We obviously need a better way to do it.

Once again, we are saved by a dynamic programming solution. But first, we rewrite $E_{p(\boldsymbol{Y}|\boldsymbol{x}^{(k)})}[F_j(\boldsymbol{x}^{(k)}, \boldsymbol{Y})]$ by moving the probability inside the implicit sum of F_j:

$$\begin{aligned} E_{p(\boldsymbol{Y}|\boldsymbol{x}^{(k)})}[F_j(\boldsymbol{x}^{(k)}, \boldsymbol{Y})] &= \sum_{\boldsymbol{y} \in \mathcal{Q}^T} p(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{x}^{(k)}) F_j(\boldsymbol{x}^{(k)}, \boldsymbol{y}) \\ &= \sum_{i=1}^{T} \sum_{q', q \in \mathcal{Q}^2} p(\boldsymbol{Y}_{i-1} = q', \boldsymbol{Y}_i = q|\boldsymbol{x}) f_j(q', q, \boldsymbol{x}, i) \end{aligned}$$

This new expression is easier to live with, since $p(\boldsymbol{Y}_{i-1} = q', \boldsymbol{Y}_i = q|\boldsymbol{x})$ can be computed with a dynamic programming approach. We do this in the same

way as the Forward algorithm for HMMs, which computes the probability of a given state at any point in the input, given the previous observations. But since the dependencies of the CRF are undirected, we also need to find the probability of the remaining words, given the state q at time t, known as the backward.

We denote the forward probability of q at time t, given the input $\boldsymbol{x}$ as $\alpha_t(q|\boldsymbol{x})$ and the backward as $\beta_t(q|\boldsymbol{x})$, which gives the following expression for $p_\theta(\boldsymbol{Y}_{i-1} = q', \boldsymbol{Y}_i = q|\boldsymbol{x})$:

$$p(\boldsymbol{Y}_{i-1} = q', \boldsymbol{Y}_i = q|\boldsymbol{x}) = \frac{\alpha_{i-1}(q'|\boldsymbol{x}) M_i(q', q|\boldsymbol{x}) \beta_i(q|\boldsymbol{x})}{Z(\boldsymbol{x})} \tag{5.4}$$

using the following expressions to calculate the forward and backward:

$$\begin{aligned}
\alpha_0(q|\boldsymbol{x}) &= \begin{cases} 1 & \text{if } q = \mathsf{start} \\ 0 & \text{otherwise} \end{cases} \\
\alpha_t(q|\boldsymbol{x}) &= \sum_{q' \in \mathcal{Q}} \alpha_{t-1}(q'|\boldsymbol{x}) M_t(q', q|\boldsymbol{x}) \qquad 1 < t \leq T+1 \\
\beta_{T+1}(q|\boldsymbol{x}) &= \begin{cases} 1 & \text{if } q = \mathsf{stop} \\ 0 & \text{otherwise} \end{cases} \\
\beta_t(q'|\boldsymbol{x}) &= \sum_{q \in \mathcal{Q}} \beta_{t+1}(q|\boldsymbol{x}) M_{t+1}(q', q|\boldsymbol{x}) \quad 1 \leq t \leq T
\end{aligned}$$

These relations give Viterbi-like algorithms using a trellis to store intermediary computations in the same way that equation (4.3) resulted in the Viterbi algorithm. Lafferty, McCallum and Pereira (2001) formulate the forward and backward recursive relations as (row) vector valued functions $\alpha_t(\boldsymbol{x}) = \alpha_{t-1}(\boldsymbol{x}) M_i(\boldsymbol{x})$ and $\beta_t(\boldsymbol{x})^\top = M_{t+1}(\boldsymbol{x}) \beta_{t+1}(\boldsymbol{x})^\top$, but this is just a shorthand that avoids the explicit sums in the expressions above.

5.4 Practical matters

As stated above in section 5.2, the feature weights θ_i have to be numerically approximated. This is a quite costly operation, especially for large tagsets, since for each feature we need to perform a $O(|\mathcal{Q}|^2 T)$ operation for each sentence in the training set; this means that a doubling of the tagset size will give a fourfold increase in training time. Most likely it will increase training time even further, since an additional number of tags will yield a larger number of features as well.

We used wapiti, a linear-chain CRF toolkit described in Lavergne, Cappé and Yvon (2010) to perform the experiments described in chapter 6, and it supports a number of approximation algorithms and regularisation options. The regularisation parameters are penalties applied to the likelihood function, and serve to mitigate the tendency of log-linear to overfit the training

data. We did not explore the possibilities of wapiti's regularisation options. However the choice of algorithms and how to combine them was explored to some extent, since this can have a large impact on how long it takes to train a model.

The algorithm that yields the best models is L-BFGS. But while it gives good results when done, it can take a long time to get to that result. For this reason, we had the best results with first running a fixed number of iterations of the Rprop (*Resilient backpropagation*) algorithm first; 10 to 20 iterations gave the best results, the more iterations the bigger the model. Rprop is a purely first-order algorithm (in contrast with L-BFGS which incorporates second-order information) which simply finds the direction of the maximum using the gradient and takes a big step in that direction. This means that the algorithm gets to the general neighbourhood of the maximum fairly quickly. After the Rprop iterations were done, we then ran L-BFGS to convergence to get as good a model as possible.

Chapter 6

Experiments & evaluation

With the underlying theory of HMMs and CRFs established, we can proceed to evaluating them as models of Latin. As outlined in the introduction, there are two distinct tagging tasks that have to be solved: MSD tagging and PoS tagging. For this reason we will present four series of experiments: MSD and PoS tagging using HMMs, and MSD and PoS tagging with CRFs. In each series of experiments, the experiments can be put in one of two groups. The first type of experiment serves to measure the performance of the models on the same kind of data as they were trained on, the second to get a rough idea of how different the two corpora available (*BG* and *Vulgata*) are. For all the experiments we computed four different metrics: overall tagging error (TE), the fraction of sentences with at least one mistagged token (SE), and tagging error for out-of-vocabulary (OOV) and in-vocabulary (IV) tokens.

The first set of experiments are 10-fold cross validation experiments. We split the corpus into 10 non-overlapping parts (the folds), and then run 10 experiments, each using a different fold for testing and the remaining nine for training the model. The final score for the full experiment is the average of the scores from the 10 sub-experiments. There are several ways to split the corpus, but we have used a round-robin approach that places the first sentence in the first fold, the second in the second fold, and so on up to sentence ten (which is placed in the tenth fold, of course). Then we then start over, placing sentence eleven in the first fold, number twelve in the second, and so on[1].

In the second set of experiments, we used either *BG* or *Vulgata* to train a model and used the other corpus for testing. As mentioned above, the purpose of these experiments is to get an idea of how different the *BG* and *Vulgata* corpora are. The out-of-domain effect is well documented in the literature, both for tagging (Poudat and Longrée 2009) and parsing (Gildea 2001), and we expect the difference between testing on in-domain data and out-of-

[1] For the mathematically inclined, sentence n is placed in fold $k \equiv n \pmod{10}$, or fold $F_i = \{s_n | n = i + k10, k \in \mathbb{N}\}$.

domain data to be large. The experiments are labeled "*X* on *Y*" where *X* is the corpus used for training and *Y* the test corpus.

Finally, a note about hypothesis testing, a more rigorous way of comparing experimental results than simply comparing averages. In the context of our 10-fold experiments, this means that we see the results of the individual 10 experiments as drawn from some random distribution with certain properties. We can then compute the probability that the results of two full experiments are drawn from the same distribution. There are several ways to do this, but we use *Student's t-test*, a relatively simple and robust test. (Woods, Fletcher and Hughes 1986, 176–181)

The underlying assumption of the t-test is that both populations (the results of the two experiments) are normally distributed and have the same variance. We estimate the medians $\bar{X}_1$ and $\bar{X}_2$ of the populations as the average of the individual results, and the variances $s_i^2 = \sum_{j=1}^{n_i} (\bar{X}_i - x_j)^2 / n-1$. We compute the test statistic

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{s^2/n_1 + s^2/n_2}}$$

$$s^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$$

and then compare t with a table or computer program and find the probability that the two populations have the same mean; if this p is sufficiently small (usually smaller than 0.05 or 0.01) we judge the difference to be *significant*. (Woods, Fletcher and Hughes 1986, 176–181)

6.1 HMM MSD tagging

TnT can optionally encode capitalisation as part of the tagset, which is reported to increase accuracy for English (Brants 2000). While not obviously a useful feature for the Latin data, since proper names, the only words that are capitalised, get the same MSD tags as common nouns, we ran two series of experiments: with and without capitalisation. Table 6.1 gives the results for the models without capitalisation and table 6.2 the results with capitalisation. The results do not appear significantly different from each other, an impression that is upheld by the T-test ($p > 0.05$ for all the experiments and error rates). For this reason we will use the data in table 6.1 in our discussion of the results.

As the numbers make clear, the *BG* tagging task is quite a bit harder than the *Vulgata*, with 57% increase in tagging error, 78% in sequence error and 39% increase in in-vocabulary error. Interestingly, the OOV error is only 10% larger, probably because of TnT's cap on the occurence counts of the suffixes used to handle unknown words. Most of this difference is likely due to the important differences in corpus between the two corpora: the *Vulgata* is four

Experiment	TE	SE	OOV	IV
10-fold joint	11.1 %	52.4%	36.0%	8.98%
10-fold *BG*	15.7 %	86.5%	39.3%	11.1 %
10-fold *Vulgata*	9.98%	48.6%	35.7%	7.97%
BG on *Vulgata*[a]	37.2 %	92.6%	66.7%	15.0 %
Vulgata on *BG*[b]	30.1 %	97.2%	51.6%	17.6 %

[a] 235 unobserved tags in test data
[b] 42 unobserved tags in test data

Table 6.1: HMM MSD experiments, no capitalisation

Experiment	TE	SE	OOV	IV
10-fold joint	11.1 %	52.3%	35.7%	8.99%
10-fold *BG*	15.6 %	85.7%	38.5%	11.1 %
10-fold *Vulgata*	9.98%	48.5%	35.7%	7.98%
BG on *Vulgata*	37.5 %	92.9%	67.1%	15.2 %
Vulgata on *BG*	30.3 %	96.9%	52.1%	17.6 %

Table 6.2: HMM MSD experiments, with capitalisation

and a half times the size of the *BG* corpus. The joint corpus, while bigger, gives slightly worse results than using the *Vulgata* on its own. We attribute this difference to this corpus being less uniform than the plain *Vulgata* corpus, which still accounts for 82% of the combined corpus.

The overall error rates for the out-of-domain are not surprising, and *BG* gives about 25% more errors on *Vulgata* than vice versa. We can point to several reasons for this. Again, the *BG* corpus is vastly smaller than its Biblical counterpart; also, the effect of 235 tags in the *Vulgata* not occuring in *BG* should not be underestimated. But if we compare the tagging errors with the results on in-domain data, a different picture emerges. The transition to a new domain has tripled the error for the *Vulgata*-trained model, while the *BG* model's error has increased by a factor of 2.3. This could indicate that the *BG* is a better model of Biblical Latin than the *Vulgata* is a model for Classical Latin, but that the overall results are worse for the *BG* due to a lack of data.

To better judge the *Vulgata* as a model of *BG*, we decided to run some additional experiments, using a subset of the *Vulgata* for training and testing that on the *BG*. For our subset we decided to combine gospels so that their size was roughly the same as the *BG* corpus. The combinations of gospels that fit the size of the *BG* corpus best were *Mark* and *John* (24,595 tokens) and *Mark* and *Matthew* (27,095 tokens). While *Mark* and *John* is a better fit in terms of size, the combined corpus may be less uniform since the gospel of

Experiment	TE	SE	OOV	IV
Mark & *Matthew*[a]	39.1%	99.1%	58.4%	18.5%
Mark & *John*[b]	39.2%	98.9%	58.3%	17.9%

[a] 102 unobserved tags in test data
[b] 123 unobserved tags in test data

Table 6.3: Small HMM *Vulgata* experiments

John is the one non-synoptic[2] gospel. Therefore we tested *Mark* and *Matthew* as well as *Mark* and *John* in case the combination of synoptic gospels turns out to be markedly different.

The results of the small *Vulgata* experiments are summarised in table 6.3. The two variants turned out to not give very different results, and as hypothesised, a smaller section of the *Vulgata* performs worse than the *BG* on out-of-domain data. The difference in overall and in-vocabulary error is reasonably small at only two and three percentage points, respectively, but sentence error goes up more than six points. On the other hand, OOV error drops by almost eight and a half points; one possible explanation for this is that the assumptions of TnT's OOV model, using only the suffixes of rare words to predict suffixes of unseen words, fits these experiments better than the larger experiments. We will explore the possible issues with TnT's OOV model and Latin further in the following sections.

Poudat and Longrée (2009) report results of tagging Latin using models trained with TnT on data from the Latin part of the LASLA corpus[3]. Unfortunately, the raw data of the corpus are not avilable (Haug et al. 2009), which is why this corpus has not been discussed earlier. Using books 1–2 and 4–7 for training and book 3 for testing, a tagging accuracy of 84.26% is reported, which is perfectly in line with the results from our 10-fold *BG* experiment. The reported size of the tagset is 3732 tags, but it is unclear how many distinct tags occur in the *BG* part of the corpus.

Another point of comparison is the learning curve reported in figure 3 of Brants (2000), which gives an overall accuracy of about 95% by 10-fold cross validation on 100,000 tokens of the German language NEGRA corpus, an OOV accuracy of about 86%, and IV around 97%. The results on the PROIEL corpus are not quite on par with this. Best was the 10-fold *Vulgata* experiment, which gave an overall accuracy of 90.0%, OOV accuracy of 64.3%, and IV accuracy of 92.0%. We attribute this difference in accuracy to the size of the PROIEL tagset, with 544 distinct tags in the *Vulgata* part of the corpus, compared to the more modest 57 tags of the NEGRA corpus.

[2] The synoptic gospels, *Mark*, *Matthew* and *Luke*, contain roughly the same stories in roughly the same order, and to a large extent with the same wording. *John* is different in structure and narrative.

[3] `http://www.cipl.ulg.ac.be/Lasla/`

Experiment	TE	SE	OOV	IV
10-fold joint	3.91%	26.0%	19.2%	2.62%
10-fold *BG*	7.12%	63.7%	21.9%	4.21%
10-fold *Vulgata*	3.36%	22.8%	19.9%	2.08%
BG on *Vulgata*	23.8 %	81.2%	47.3%	6.09%
Vulgata on *BG*	16.6 %	88.2%	32.5%	7.33%

Table 6.4: HMM full PoS tagger

Experiment	TE	SE	OOV	IV
10-fold joint	3.13%	22.2%	16.9%	1.97%
10-fold *BG*	5.56%	56.8%	19.1%	2.89%
10-fold *Vulgata*	2.73%	19.5%	17.6%	1.57%
BG on *Vulgata*	19.6 %	76.2%	41.1%	3.53%
Vulgata on *BG*	13.7 %	84.3%	27.5%	5.58%

Table 6.5: HMM major PoS tagger

For the 10-fold *BG* experiment the results are even less cheerful. For 20,000 tokens of NEGRA, Brants (2000) gives an overall accuracy of about 93%, OOV accuracy a bit above 80%, and IV accuracy at about 96%; compared to 84.3% overall, 60.7% OOV accuracy, and 88.9% IV accuracy on the *BG* corpus. Again, this is likely due to the size of the tagset (344 tags in the *BG* corpus) and the relatively free word-order of Classical Latin.

6.2 HMM PoS tagging

A complete tagging solution for Latin according to the PROIEL tagset should also do PoS tagging. Again, we performed two sets of experiments; a first series using the full PoS tagset (see table 2.14) and a second with only the major field of the tag. In the full tagset the tag `Ne` is used on proper nouns; as mentioned above, proper nouns are capitalised in the corpus, which might mean that the capitalisation option of TnT will boost performance on the full PoS task. However models trained with this option did not significantly outperform those without it. Therefore, the numbers reported are for models without the capitalisation option. Tables 6.4 and 6.5 give the results on the full and major PoS tagsets, respectively.

Since the size of the tagset has been reduced by more than a full order of magnitude compared to the MSD tagging experiments, it should come as no surprise that the PoS tagging results are quite a bit better. But apart from the fact that the numbers are better, the PoS and MSD experiments behave in a similar way. The pure Vulgata corpus gives the best results, with the less uni-

Experiment		TE	SE	OOV	IV
Mark & *Matthew*,	full PoS	24.1%	94.3%	38.4%	8.88%
Mark & *John*,	full PoS	25.5%	94.8%	40.7%	8.49%
Mark & *Matthew*,	major PoS	20.4%	92.5%	33.9%	5.98%
Mark & *John*,	major PoS	22.7%	93.9%	37.5%	6.11%

Table 6.6: Small HMM *Vulgata* PoS experiments

form joint corpus lagging a bit, but not much. The *BG* lags behind the other two, more than in the MSD experiments, with roughly twice the amount of incorrect tokens and three times as many incorrect sentences. OOV error is much less affected by the more limited training data, just like for the MSD experiments, increasing just 10% in the *BG* experiments. Apart from a performance boost due to the even smaller tagset, the major PoS tag experiments give similar results as for the major tag.

An interesting detail is the OOV metric of the joint and *Vulgata* experiments. Here, the joint corpus gives a better OOV error than the *Vulgata* model by about three quarters of a percentage point. This difference is not significant ($p > 0.05$) however.

Moving on to the out-of-domain experiments, the difference between the two experiments looks more in line with the in-domain data. The model trained on the *BG* gives almost twice the overall error rate as the one trained on *Vulgata*; sentence and in-vocabulary error is quite a bit better with the *BG* however, and it's OOV error that gives the worse overall result. The smaller *Vulgata* selections give much the same results as for the MSD experiments though. Both in-vocabulary and sentence error increase by several percentage points, but the drop in OOV error makes the overall error rates differ by only a few points. Again, this drop in OOV error is probably due to the differences between *Vulgata* and *BG* being more in line with the assumptions underlying TnT's OOV model.

The tagsets in the PoS tagging experiments are more in line with the tagset used in the NEGRA experiments from Brants (2000), as are the results. In fact, the overall and in-vocabulary errors of the *Vulgata* experiment with the full PoS tag are comparable to the best values (2.3% and 3.3% error, respectively) reported for 320,000 tokens. This is probably due to the smaller size of the tagset, which is roughly half the size of the NEGRA tagset.

On the other hand, the OOV error is a lot worse. In fact, the learning curve in Brants (2000) crosses the 80% accuracy line somewhere between 10 and 20 thousand training tokens. This is probably due to the mismatch between TnT's OOV model and the facts of Latin. As stated in 4.3, only suffixes of words occuring less than 10 times in the corpus are used, based on the assumption that unseen words are probably rare words, and thus only in-

No.	Feature	Description
1	$w_t = w \wedge q_t = q$	word surface form & label
2	$q_t = q$	unigram label
3	$q_{t-1} = q' \wedge q_t = q$	bigram label pair
4	$m_t = i \wedge q_t = q$	morphological suffix & label
5	$m_t = i \wedge q_{t-1} = q' \wedge q_t = q$	morphological suffix & label pair
6	$s_{n,t} \wedge q_t = q$	n letter suffix & label
7	$p_t = p \wedge q_t = q$	PoS tag & label
8	$p_t = p \wedge q_{t-1} = q' \wedge q_t = q$	PoS tag & label pair

Table 6.7: CRF feature templates

frequent words should be used to estimate the emission probabilities. Latin however, is a richly inflecting language, and it's entirely reasonable to think that a new word is simply a form of a word that happened not to crop up in training, rather than a word that is rare in and of itself.

6.3 CRF Feature selection

The possibilities for features are almost unbounded with CRFs, but for our experiments we've kept it fairly simple. wapiti has a simple template system that generates binary feature function from patterns observed in training. If the pattern is seen in the sequence to be classified, the function's value is 1, and 0 in any other case.

Table 6.7 shows the templates used in our experiments. There are two broad classes of feature used: lexical features, directly derivable from the surface form of the word, and PoS tag based features. The PoS tag is obviously a valuable feature, but has the not insignificant drawback of not being directly derivable from running text. Thus, running text will have to be PoS tagged first and the output of that tagger fed to the MSD tagger as a feature. We'll explore this further in section 6.6, but not before we explore these tagging problems without the additional complication of layered taggers. The base model will be templates 1 through 3, and we'll explore combinations of the other features to gauge the potential of CRFs. We use template 6 as a simple emulation of TnT's OOV model but without the frequency cutoff, using templates for suffixes of length 1 to 10.

6.4 CRF MSD tagging

The general structure of the CRF experiments is the same as for the HMMs, but there are two differences: First, the number of possible experiments is much greater, since we can vary the features more or less as we please. Sec-

Experiment	TE	SE	OOV	IV
HMM 10-fold *BG*	15.7 %	86.5%	39.3%	11.1 %
10-fold BG^a	26.3 %	95.6%	93.3%	12.2 %
10-fold BG^b	18.7 %	91.0%	44.5%	13.2 %
10-fold BG^c	23.8 %	94.6%	62.7%	15.6 %
10-fold BG^d	16.4 %	88.4%	37.2%	12.0 %
HMM 10-fold *Vulgata*	9.98%	48.6%	35.7%	7.97%
10-fold $Vulgata^b$	11.5 %	53.0%	44.8%	8.72%
10-fold $Vulgata^c$	13.4 %	57.8%	57.9%	9.69%
10-fold $Vulgata^d$	10.3 %	49.5%	34.5%	8.22%

Feature sets:
[a] 1–3 (base model)
[b] 1–4
[c] 1–3, 5
[d] 1–3, 6 ($1 \leq n \leq 10$)

Table 6.8: CRF MSD experiments, lexical features

ond, training a model takes several orders of magnitude longer; with TnT, training a model with the full *Vulgata* corpus takes about half a second. Training a model on the *BG* corpus, using sixteen parallel threads of computation, takes ten to fifteen minutes, depending on how quickly the approximation algorithms converge. For this reason, the majority of experiments have been with the *BG* corpus, since this let us run several 10-fold cross-validation experiments per day, as opposed to the full *Vulgata* or joint experiments, where a full 10-fold run takes approximately five or six hours.

Finally, when training the CRF models, we use a development corpus in addition to the test and training sets we used for HMM training. In this scheme, the feature functions are generated from the training set, while the feature weights θ_k are approximated so that they maximise the likelihood of the development set, rather than the likelihood of the training set. This is done because a model whose feature weights are optimised for the training data will tend to overfit and generalise badly to new data. In our experiments, we used the fold before the test fold as the development set.

We have divided our experiments into two batches. First, we have the experiments using only the lexical features, summarised in table 6.8. Not surprisingly, the base model has abysmal overall performance, almost twice the error of the TnT model. Adding the morphological suffix (model *b*) gives a nice boost in performance, and halves OOV error. Making the morphological suffix feature a bigram rather than unigram (model *c*) results in the model overfitting the training data. The best of the lexical-only models is model *d*, the base model with the 10-letter suffix features, which is about as good as

Experiment	TE	SE	OOV	IV
HMM 10-fold *BG*	15.7 %	86.5%	39.3%	11.1 %
10-fold BG^a	12.9 %	81.4%	26.3%	10.1 %
10-fold BG^b	13.4 %	82.4%	27.1%	10.5 %
10-fold BG^c	15.2 %	86.7%	33.2%	11.4 %
10-fold BG^d	13.0 %	80.9%	26.4%	10.2 %
HMM 10-fold *Vulgata*	9.98%	48.6%	35.7%	7.97%
10-fold $Vulgata^a$	8.73%	44.8%	24.8%	7.37%
10-fold $Vulgata^c$	9.59%	47.6%	28.7%	7.98%
10-fold $Vulgata^d$	8.73%	44.7%	24.4%	7.41%

Feature sets:
[a] 1–3, 6 ($1 \leq n \leq 10$), 7, full PoS
[b] 1–3, 6 ($1 \leq n \leq 10$), 7, major PoS
[c] 1–3, 6 ($1 \leq n \leq 10$), 8, full PoS
[d] 1–3, 6 ($1 \leq n \leq 10$), 7–8, full PoS

Table 6.9: CRF MSD experiments, PoS features

the TnT models. The OOV error in particular is a bit lower with this model, but the only significantly different ($p < 0.05$) error rate is the in-vocabulary error of the *BG* experiment.

Given that the *Vulgata* corpus is four and a half times as large as the *BG* corpus, one might expect that the bigram morphological suffix feature wouldn't suffer from the overfitting we saw with the *BG* model, but that turns out not to be the case. The reduction in performance is less severe with the larger corpus, but the model still overfits.

Adding the PoS tag gives a very nice boost to performance, as summarised in table 6.9, and the best CRF models now handily best the performance of TnT, and all four error rates for both experiments with model *d* are significantly different ($p < 0.05$) from the HMM models. Using the less fine-grained major PoS tag instead of the full tag increases error somewhat; the reason for this is not immediately obvious, but the fine-grained subdivision of pronouns is the most likely candidate, and might help choosing between tags with and without number. As with the purely lexical features, keeping it simple is the best approach with the PoS-based features as well, and the *Vulgata* corpus is plagued by the same overfitting as the smaller *BG*, even though the PoS features are even more compact than the inflectional suffixes.

To avoid an even further proliferation of experiments and numbers, we ran the out-of-domain experiments using only the best performing feature set (base model with suffix features and unigram feature on the full PoS tag); the results are summarised in table 6.10. In relative terms, the two primary CRF out-of-domain experiments perform about the same as their HMM cousins,

Experiment	TE	SE	OOV	IV
BG on *Vulgata*	30.2%	86.3%	50.6%	14.4%
Vulgata on *BG*	23.3%	94.4%	35.5%	15.9%
Mark & *Matthew*	30.6%	96.3%	42.9%	17.0%
Mark & *John*	30.9%	96.2%	42.3%	17.8%

Table 6.10: CRF MSD out-of-domain experiments

but since the best CRF models are better, the absolute difference between the HMM and CRF out-of-domain experiments is quite large. The smaller *Vulgata* experiments seem to do a bit better than the smaller HMM experiments, but the differences are small enough that we should be careful drawing definite conclusions based on this data alone.

6.5 CRF PoS tagging

The most important difference between MSD and PoS tagging with CRFs is that training the models takes a lot less time. Apart from that, the results of the experiments are in line with what can be expected, given the results of the previous HMM and CRF experiments. For obvious reasons, we can only use lexical features in our PoS models, and we used the same feature templates as in the lexical-only MSD experiments in table 6.8. Results of the full and major PoS tagger experiments are given in tables 6.11 and 6.12, respectively.

The CRF PoS taggers compare to their HMM cousins much in the same way the CRF MSD models do to the HMM MSD models. The best model is the base model augmented with the TnT-like suffix features, which is almost but not quite as good as the corresponding TnT model; the relative performances of the different CRF models is pretty much the same as the MSD experiments as well. The TnT-like model *d* is clearly the best, with the morphological unigram feature not quite as good, and the bigram version overtraining, both for the smaller *BG* corpus and the big *Vulgata* corpus. For the PoS taggers, most of the differences between the CRF and TnT models are significant, with $p < 0.05$; only the OOV errors of the *BG* major PoS tagger and both full PoS taggers are not significantly different.

The performance of the CRF models on out-of-domain data is given in table 6.13. The full PoS tagger performs at about the same level as the corresponding TnT models on foreign data, but with slightly elevated errors. The major PoS tagger scores slightly better in overall error rate, thanks to improved OOV error rates which outweigh the slight increases in in-vocabulary error. However, in all these experiments the differences between HMM and TnT-like CRF are so small that it is hard to draw solid conclusions in the question of whether HMMs or CRFs are the better models in this case.

Experiment	TE	SE	OOV	IV
HMM 10-fold *BG*	7.12%	63.7%	21.9%	4.21%
10-fold *BG*[b]	9.61%	73.4%	28.5%	5.64%
10-fold *BG*[c]	11.2 %	77.8%	32.3%	6.72%
10-fold *BG*[d]	8.03%	68.5%	23.0%	4.88%
HMM 10-fold *Vulgata*	3.36%	22.8%	19.9%	2.08%
10-fold *Vulgata*[b]	4.18%	27.2%	25.3%	2.40%
10-fold *Vulgata*[c]	4.29%	27.3%	25.4%	2.51%
10-fold *Vulgata*[d]	3.75%	24.9%	19.7%	2.41%

Features as in table 6.8.

Table 6.11: CRF full PoS tagger

Experiment	TE	SE	OOV	IV
HMM 10-fold *BG*	5.56%	56.8%	19.1%	2.89%
10-fold *BG*[b]	7.92%	68.0%	26.1%	4.11%
10-fold *BG*[c]	8.83%	70.8%	29.3%	4.53%
10-fold *BG*[d]	6.26%	61.9%	20.2%	3.33%
HMM 10-fold *Vulgata*	2.73%	19.5%	17.6%	1.57%
10-fold *Vulgata*[b]	4.85%	29.8%	33.6%	2.43%
10-fold *Vulgata*[c]	4.81%	29.8%	33.0%	2.44%
10-fold *Vulgata*[d]	3.73%	24.9%	19.7%	2.39%

Features as in table 6.8.

Table 6.12: CRF major PoS tagger

Experiment	TE	SE	OOV	IV
BG on *Vulgata*[a]	24.5%	83.0%	47.6%	6.53%
Vulgata on *BG*[a]	17.1%	88.3%	31.4%	8.37%
Mark & *Matthew*[a]	24.9%	93.8%	37.5%	11.0 %
Mark & *John*[a]	26.3%	94.7%	39.4%	11.0 %
BG on *Vulgata*[b]	19.6%	75.9%	38.6%	4.82%
Vulgata on *BG*[b]	12.8%	81.4%	23.3%	6.36%
Mark & *Matthew*[b]	19.5%	89.7%	30.2%	7.76%
Mark & *John*[b]	21.3%	92.3%	33.2%	7.66%

[a] Full PoS tag
[b] Major PoS tag

Table 6.13: CRF PoS out-of-domain experiments

Experiment	TE	SE	OOV	IV
10-fold *BG*	16.4%	87.5%	37.1%	12.0 %
10-fold *Vulgata*	10.2%	49.3%	34.7%	8.12%

Table 6.14: Layered CRF experiments

6.6 Layered CRF

With the efficacy of the PoS tag as a CRF feature well established, we also have to evaluate how well it will work in practice. Since the PoS tag is not directly available from raw text, we first have to PoS tag the text before doing the actual MSD tagging. This first tagging step will have a certain amount of errors, which will have an impact on performance relative to the numbers from section 6.4.

For the experiments reported in table 6.14 we reused the models from the single CRF experiments, and since TnT gave superior PoS tagging results, those models were reused for the PoS step. Unfortunately, the results aren't stellar. In both cases, the error rates drop below that of the TnT models, and are indistinguishable from the lexical-only experiments, and it seems the gold PoS-trained models are very sensitive indeed to incorrect PoS tags. The *BG* error rates, with the exception of the sentence error, are all significantly ($p < 0.05$) different from the TnT results, but none of the *Vulgata* results are significant at the same p-level.

A better way to do this, in theory, is to first train the PoS model on one part of the corpus and then use that model to tag another part of the corpus that is used as training data for the MSD model. This way the MSD PoS feature weights should be adjusted in training so that unreliable tags get lower weights than when trained on gold tags. Unfortunately, this means that the amount of training data is halved, not necessarily the best thing given the scarcity of resources for Latin. Initial experiments with this approach did not improve over the results appreciably from those already reported, and the idea was not explored any further.

Chapter 7

Straitjacketed decoding

A large full-form dictionary of morphological analyses is available from the Perseus project. It would be a shame to let this resource go to waste, so we would like to see if we can use this resource to guide the decoding process of our statistical models to boost performance a bit.

All that needs to be introduced before we present constrained decoding is our representation of the constraints involved. In addition to an observed sequence of words and the parameters of the model to be decoded, the decoding process will now use an additional piece of information: the constraint function $c : W \to \mathcal{Q}^*$ that returns a set of licenced tags for all words in the vocabulary.

7.1 Theory

Before moving on to the actual implementation and experiments, we should stop and make sure that our goals are well-defined and perhaps more importantly, that they don't entail unintended consequences such as changing the probabilities of the label sequences.

While our implementation of constrained decoding is for CRFs, the theory is simpler and clearer if we consider it in the case of HMM decoding. In essence, what we want is to find not the best-scoring label sequence, but the best-scoring label sequence that conforms to the constraints given. The naïve approach would be to walk the list of possible tag sequences for our input, in decreasing order of probability, until we find one that conforms to the constraints specified.

Of course, this approach is computationally undesirable, and a better approach is indeed available: We modify how we calculate the δ_t and corresponding q_t. Instead of considering all possible combinations states, we only consider those licenced by the constraints specified. Adapted from equations

(4.3) and (4.4) we get:

$$\delta_t(q) = \begin{cases} t(q_s, q)e(q, w_1) & t = 1, q \in c(w_1) \\ \max_{q' \in c(w_{t-1})} \delta_{t-1}(q')t(q', q)e(q, w_t) & t > 1, q \in c(w_t) \end{cases} \tag{7.1}$$

$$q_t = \begin{cases} \operatorname{argmax}_{q \in c(w_T)} \delta_T(q)t(q, q_e) & t = T \\ \operatorname{argmax}_{q \in c(w_t)} \delta_t(q)t(q, q_{t+1}) & t < T \end{cases} \tag{7.2}$$

which in turn results in the pseudo-code in figure 7.1.

Input: Input sequence $w_1 \cdots w_T$
Input: Constraint function $c : W \to \mathcal{Q}^*$
Data: trellis, a $|\mathcal{Q}| \times T$ matrix
`# Initialisation`
for $q \in c(w_1)$ **do**
  $\mathsf{trellis}_{q,1} = (q_s, e(w_1, q) * t(q_s, q))$
end
for $2 \leq i \leq T$ **do**
  for $q \in c(w_i)$ **do**
    $p = \max_{q' \in c(w_{i-1})} p(\mathsf{trellis}_{q',i-1}) * t(q', q) * e(w_i, q)$
    $q' = \operatorname{argmax}_{q' \in c(w_{i-1})} p(\mathsf{trellis}_{q',i-1}) * t(q', q)$
    $\mathsf{trellis}_{q,i} = (q', p)$
  end
end
$q' = \operatorname{argmax}_{q' \in c(w_T)} p(\mathsf{trellis}_{q',T}) * t(q', q_e)$
return *The most likely path through the trellis by following the back pointers, starting with* q'

Figure 7.1: Viterbi's algorithm with constraints

As we see, the method of computing the probabilities is unchanged from the unconstrained algorithm; all that has changed is which sequences of output nodes are considered. This means that the ordering of the label sequences and their probabilities are unchanged, and in turn that this approach is equivalent to walking a list of the n best sequences and selecting the first sequence conforming to the constraints.

7.2 Converting the Perseus data

At the most superficial level, the Perseus morphological data agree with the PROIEL tagset. They use the same morphological parameters, and the parameters have the same values, with the exception of the gender parameter, where PROIEL has four extra tags to encode ambiguous gender. But going into greater detail, the waters are muddied quite a bit. Even though they agree on the fields and their values, the two sources don't agree on which

fields and values to use in various cases, which means that we have to convert the Perseus database to the PROIEL annotation standard before we can use it to constrain our language models.

There are 710,620 analyses in the Perseus database, so manually controlling and correcting the entire DB is completely out of the question. As a measure of how well (or not) we have converted the data, we defined a simple metric. For each unique wordform in the corpus which is also given one or more analyses in the Perseus data, we extract all the different MSD tags that have been assigned to that word. Each of the MSD tags from the corpus that is not in the list of tags for that word is considered to be *missing*. The fewer missing tags, the better the data have been converted. Before any changes are made, 3,588 out of 18,313 distinct analyses in the corpus are missing.

The Perseus data are distributed as an XML file containing the analyses. Since working with 140 MB of XML takes a long time, and is on the whole unpleasant, the first step in the conversion process is inserting the whole thing into a database. This also lets us use SQL for the conversion, which simplifies certain transformations considerably. Some discrepancies are fixed by simply updating the rows according to some criteria, while others are fixed by creating additional database records from the ones already present. After conversion, the number of missing tags is reduced to 2,019 and the number of rows in the DB increased to 911,187.

7.3 Implementation

The CRF toolkit we used as a base for implementing constrained decoding, wapiti, is implemented in C. Being written in C does not always mean that modification will be easy or pleasant, but wapiti's source code is clear and well-documented, which facilitated hacking, and once armed with an understanding of the underlying theory of CRFs and the proper way to constrain the decoding process, the actual implementation was relatively straightforward.

First we added awareness of constraints to the data model, by adding the necessary field to the sequence data type. Then, we needed to get the application to actually read a constrained corpus. The best way to represent the constraints isn't obvious, but in the end we decided on a simple, if inelegant, approach. A special token (|) separates the "normal" input and constraints, and a new post-processing step converts the raw input to the internal format. Any constraints that are unknown labels are silently ignored. Finally the actual constrained decoding process was implemented, as outlined in figure 7.1 with the necessary modifications for CRF decoding.

Beyond making sure that there were no catastrophic failures during the

Experiment	TE	SE	OOV	IV
10-fold BG^a	21.6 %	93.2%	37.2%	18.3 %
10-fold BG^b	20.0 %	92.2%	27.7%	18.4 %
10-fold BG^c	18.0 %	89.7%	23.2%	16.9 %
10-fold $Vulgata^b$	10.7 %	52.2%	20.6%	9.84%
10-fold $Vulgata^c$	9.87%	49.9%	16.2%	9.33%

Feature sets:
[a] Table 6.8 [a]
[b] Table 6.8 [d]
[c] Table 6.9 [a]

Table 7.1: Constrained CRF experiments

constrained decoding experiments, we used valgrind[1] to check for memory leaks and other memory-related problems that can crop up when programming in C. valgrind is an open-source toolkit (available for Linux and OS X 10.5 and 10.6) that intercepts all memory accesses and heap manipulations (`malloc`/`free`) to make sure nothing unintended happens. Reading or jumping based on data in an indeterminate state (typically an uninitialised variable or array element) is reported, as is any unfreed heap memory remaining at the end of the process, which means that a program that runs without valgrind complaining is less likely to cause problems later. Thankfully, wapiti did not trigger any of valgrind's alarms before modification, so simply making sure this property remained constant kept the constrained decoding code clean as well.

7.4 Experiments

To evaluate the constrained decoding, we took some of the models trained in section 6.4 and reran the experiments with constraints, giving the results in table 7.1. As the numbers make clear, the results aren't all good. The constrained decoding has a beneficial effect on OOV error, but IV error increases by about 6 points in the *BG* case and almost two for the *Vulgata*, both with and without the PoS tag feature. The base *BG* experiment is the only to have an overall reduction in error, which is due to the 60% reduction in OOV error; in-vocabulary error increases by the same six points as the other models. All the error metrics for all experiments are significantly different ($p < 0.05$) from the corresponding HMM MSD results in table 6.1, except the BG^a OOV error and $Vulgata^c$ TE score. All the error rates in table 7.1 are significantly different ($p < 0.05$) from the corresponding TnT scores, with the exception of overall and sentence error for the $Vulgata^c$ experiment. The most likely

[1] `http://valgrind.org`

Experiment	TE	SE	OOV	IV
BG on *Vulgata*	25.8%	82.9%	39.5%	14.7%
Vulgata on *BG*	27.0%	96.6%	34.8%	26.7%
Mark & *Matthew*	28.9%	97.5%	37.0%	28.6%
Mark & *John*	30.0%	97.5%	39.1%	29.6%

Table 7.2: Constrained CRF, out-of-domain

cause for this unfortunate drop in performance is the imperfect conversion of the Perseus data to the PROIEL conventions, in particular the ambiguous gender tags, which have to be induced from the Perseus data. Due to these imperfections, there is a lower bound on the error rates, 11.2% for the *BG* corpus and 4.09% on *Vulgata*.

The out-of-domain experiments, on the other hand, did not suffer. We used the best lexical-only CRF models (table 6.8 *d*) to test constrained decoding on out-of-domain data, and as the numbers in table 7.2 show, constrained decoding helps quite a bit. In-vocabulary error increases by 10 points for the *Vulgata*-trained models, but the 20-point reduction in OOV error results in a nice drop in overall error for the two smaller models, and a few for the full model. The *BG*-trained model has more or less the same IV error as its HMM cousin, but the reduction of OOV error from 67% to 40% results in the same 10-point reduction in overall error as the small *Vulgata* models.

To see how the constrained decoding procedure fares with better data for the constraints, we ran the same experiments as outlined above, but with data from the PROIEL corpus in addition to the converted Perseus data. For each token that had constraints in the converted Perseus data set, we added the possible analyses for that token that occur in the PROIEL corpus. While this certainly is cheating, it might give a more accurate view of the potential of the constrained decoding procedure. The results of the constrained decoding experiments with PROIEL data added are given in table 7.3; while the error rates are most likely artificially low, since most of the constraint sets that were missing the correct constraints with the PROIEL data are probably a bit too small with the PROIEL data, this at least removes the problem of many words missing the correct tag altogether.

Adding the PROIEL data definitely makes a difference for the in-domain experiments, with error levels dropping to the levels of the CRFs with gold PoS tags in table 6.9, but using only lexical features, and only the *BG* sentence error and *Vulgata* IV error are not significantly different ($p < 0.05$) from the HMM experiments. In-vocabulary error is a bit higher for the *BG* model, but the OOV error is reduced by 15 points for both models, which gives a nice improvement overall. For the out-of-domain experiments, the addition of the PROIEL data turned out to have less of an impact, with a reduction in error of only a few points in most cases. An interesting difference is the

Experiment	TE	SE	OOV	IV
10-fold *BG*	13.7 %	84.4%	23.0%	11.7 %
10-fold *Vulgata*	8.81%	45.5%	19.7%	7.89%
BG on *Vulgata*	24.8 %	81.7%	35.3%	24.7 %
Vulgata on *BG*	24.2 %	95.5%	34.6%	17.6 %
Mark & *Matthew*	26.9 %	96.9%	36.8%	20.7 %
Mark & *John*	28.0 %	96.7%	37.0%	22.3 %

All models using table 6.8 [d]

Table 7.3: Constrained CRF, with PROIEL

in-vocabulary error of the *BG* on *Vulgata* experiment, which increased by a full 10 points.

Chapter 8

Conclusion

We are now in a position to answer some of the more general problems posed in the introduction. As should be obvious from the preceding chapters, automatic analysis of Latin morphology is very much possible. The results, especially for Classical Latin, are still quite a distance from those reported for other languages, but so is the amount of data available. Still, the results are far from depressing, and should be useful for bootstrapping further resources.

Sequence classification with statistical models is definitely a working approach for this task, both with HMM and CRF models, and the HMM models perform on a level comparable with results reported previously in the literature, both directly comparable results on Latin (Poudat and Longrée 2009) and indirectly on other languages (Brants 2000). Perhaps less cheerful, but interesting nonetheless, is the fact that the CRF models for practical purposes perform no better than the HMMs. There are two possible explanations for this. Either the additional descriptive power of CRFs is simply not necessary and the relatively simplistic HMM model is sufficient, or the CRF model is suffering from a lack of data. A priori we lean toward the latter, as log-linear models (which a CRF is) in general require more training data than HMMs.

Not surprisingly, the TnT-like OOV model of using suffixes to handle unknown words is very useful for Latin. This is of course due to the suffix-inflecting nature of Latin. More interesting is the somewhat erratic behaviour of TnT's OOV performance on out-of-domain data. Our interpretation of this is that the underlying assumption of the frequency cutoff for words used to build the suffix model is untenable for Latin, where an unknown word may just as well be an unseen form of a common word as a rare word. This can be tested using HunPos (Halácsy, Kornai and Oravecz 2007), an open-source reimplementation of TnT, which allows greater access and control over the internal parameters which are fixed in TnT. Unfortunately, technical issues made it impossible to use HunPos.

8.1 HMM or CRF?

Given the results presented here, the choice between HMM and CRF is fairly obvious. CRFs take a long time to train and require a certain amount of work to find the best features; training HMMs is very fast, and no feature engineering is required: the corpus goes in and the model comes out. Combined with the fact that they give the same results, HMMs have a lot going for them.

However, statistical language models are highly sensitive to data from new domains, as is shown by the out-of-domain experiments in chapter 6. Thus, if the corpus to be annotated is very different from the data the model has been trained on, a CRF model with the constrained decoding process outlined in chapter 7 may very well be a better choice. Of course, this scenario is not unlikely in the intended application of accelerating treebanking of new Latin data.

The experiments performed in Poudat and Longrée (2009) can serve as an indicator of how different certain combinations of texts are. For example, using *Bellum Civile*, the other surviving work of Caesar, an account of the civil war following Caesar's crossing of the Rubicon, to train a model and testing on book 3 of the *BG*, gives an overall tagging error of 19.9%. Given that this corpus is slightly smaller than the *BG* corpus at 33,221 tokens compared to the 42,055 of books 1–2 and 4–7 of *BG*, this indicates that the two texts are very similar indeed and a constrained CRF would probably not be useful. On a text like Cicero's first Catilinarian on the other hand, where the *BG*-trained model has 39.1% error, a constrained CRF will probably be more useful.

8.2 Future work

There are a number of things that would be interesting to investigate further in connection with automatic analysis of Latin. One is the internal structure of the MSD tags. The individual tags are not atomic, they have a certain amount of internal structure that it would be interesting to exploit further. Unfortunately this is not easily done with the traditional approaches, short of training models for each individual field. In the case of CRF models, this also means we have to find the best ordering of models, since the output of one model can be used as an input feature of the models later in the pipeline. A more promising approach is Schmid and Laws's (2008) RFtagger, which trains higher order HMMs that model the fine-grained structure of this kind of tag.

Also, for a truly complete end-to-end tagging solution for Latin, we need to tackle the problem of clitics. As outlined in the end of section 2.4, clitics are combined with the preceding word and written as a single unit. This word-clitic unit can be confused with legitimate word forms in some cases, for example the token *oratione* can be read as the ablative singular of *oratio*,

or as the nominative singular with the interrogative *ne*. This means that ambiguous tokens have to be classified in some way. Initial experiments treating this problem as a sequence classification problem using CRFs have given promising results, but time did not allow this to be fully explored for the present work.

Appendix A

Multinomial MLE

When estimating the parameters of a HMM, we assume that our training corpus is the result of a *mutinomial experiment*. Such an experiment is the combined result of several *trials*, where each trial can result in one of several *outcomes*, each of which have fixed probabilities (Walpole et al. 2007). A very simple example of a multinomial experiment is drawing coloured balls from an urn and putting the drawn ball back after each draw. In this case, a trial is the drawing of a ball, and the possible outcomes are the different colours of the balls in the urn.

The multinomial distribution has the parameters n, the number of trials, and $p_1 \cdots p_k$, the probability of a trial resulting in each of the outcomes. To ensure correct probabilities, the constraint $\sum_{i=1}^{k} p_i = 1$ is required. The probability of n trials resulting in the counts $x_1 \cdots x_k$ for each of the k different outcomes is then:

$$p(x_1, \ldots, x_k; n, p_1, \ldots, p_k) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$$

when $\sum_{i=1}^{k} x_i = n$. If the number of outcomes is different from the number of trials, the probability is obviously 0.

As stated in 5.2, likelihood is a measure of the probability of a parameter set given some data. Using the assumption that our corpus is the result of a multinomial experiment, we use the probability of the training data given the parameters as our likelihood function:

$$L(N, p(q_1), \ldots, p(q_k); c(q_1), \ldots, c(q_k)) = p(c(q_1), \ldots, c(q_k); N, p(q_1), \ldots, p(q_k))$$

Where $c(q_i)$ is the number of times the tag q_i occurs, $p(q_i)$ is the corresponding tag probability, and N is the number of words in the corpus. But just like the CRF likelihood function, we choose to maximise $\mathcal{L} = \log L$ instead; the logarithm is a monotonically increasing function, which means that $\mathcal{L}$ will have the same maxima as L, and the properties of the logarithm gives a

function that is a lot easier to work with:

$$\mathcal{L} = \log L = \log N! + \sum_{i=1}^{k} c(q_i) \log p(q_i) - \sum_{i=1}^{k} \log c(q_i)!$$

However $\mathcal{L}$ does not have a global maximum, since its value can always be increased further by increasing the values of the $p(q_i)$. But they cannot be increased freely; they are constrained to always sum to 1, which means that our problem is a constrained optimisation problem, and we can apply the technique known as *Lagrange multipliers*. Without going into the gory details, this means that finding the maximal value of a function $f(\vec{x})$ with the constraint $g(\vec{x}) = 0$ can be done by solving the equation $\nabla f(\vec{x}) = \lambda \nabla g(\vec{x})$, or equivalently $\partial/\partial x_i f = \lambda \partial/\partial x_i g$ for all the dimensions x_i of the functions.

In our case of the multinomial distribution, the function to be maximised is $\mathcal{L}$ and the constraint is $g(p(q_1), \ldots, p(q_k)) = \sum_{i=1}^{k} p(q_i) - 1 = 0$. We then solve $\partial/\partial p(q_i) \mathcal{L} = \lambda \partial/\partial p(q_i) g$:

$$\begin{aligned}
& \frac{\partial}{\partial p(q_i)} \mathcal{L} = \lambda \frac{\partial}{\partial p(q_i)} g \\
\Leftrightarrow \quad & \frac{\partial}{\partial p(q_i)} \sum_{i=1}^{k} c(q_i) \log p(q_i) = \lambda \frac{\partial}{\partial p(q_i)} \sum_{i=1}^{k} p(q_i) \\
\Leftrightarrow \quad & \frac{\partial}{\partial p(q_i)} c(q_i) \log p(q_i) = \lambda \\
\Leftrightarrow \quad & \frac{c(q_i)}{p(q_i)} = \lambda \\
\Leftrightarrow \quad & p(q_i) = \frac{c(q_i)}{\lambda}
\end{aligned}$$

Now all that remains is to find the value of λ. Since we have $p(q_i) = c(q_i)/\lambda$ for all i, we can sum all the left-hand and right-hand sides and set them equal:

$$\begin{aligned}
& \sum_{i=1}^{k} p(q_i) = \sum_{i=1}^{k} \frac{c(q_i)}{\lambda} \\
\Leftrightarrow \quad & \frac{\sum_{i=1}^{k} c(q_i)}{\lambda} = 1 \\
\Leftrightarrow \quad & \frac{N}{\lambda} = 1 \\
\Leftrightarrow \quad & \lambda = N
\end{aligned}$$

and we get the maximum-likelihood estimate $\hat{p}(q_i) = c(q_i)/N$. We proceed in the same way to get the estimates $\hat{p}(q', q) = c(q', q)/N$ and $\hat{p}(q, w) = c(q, w)/N$, which give the estimates $\hat{p}(w|q) = c(w, q)/c(q)$ and $\hat{p}(q|q') = c(q', q)/c(q')$ from chapter 4 when combined when the relation $p(x|y) = p(x, y)/p(y)$.

Appendix B

Morphemes

B.1 Nominal morphemes

The base nominal morphemes are those given in table B.1. Additionally, the comparative adjective morphemes are created by appending the third declension morphemes to the comparative morpheme *–ior*, and the superlative morphemes with the first and second declension morphemes and the superlative morph *–issim*. Finally *–iter* and *–ius* take care of adverb derivation of third declension adjectives and the comparative of adverbs, respectively.

Nom.	Voc.	Acc.	Gen.	Dat.	Abl.	Description
a	a	am	ae	ae	a	1st decl. sg.
ae	ae	as	arum	is	is	1st decl. pl.
us	e	um	i	o	o	2nd decl. sg.
i	i	os	orum	is	is	2nd decl. pl.
a	a	a				2nd decl. n. pl.
–	–	em	is	i	e	3rd decl. sg.
es	es	es	us	ibus	ibus	3rd decl. pl.
a	a	a				3rd decl. n. pl.
us	us	um	us	ui	u	4th decl. sg.
us	us	us	uum	ibus	ibus	4th decl. pl.
ua	ua	ua				4th decl. n. pl.
s	s	m	s	i	–	5th decl. sg.
s	s	s	rum	bus	bus	5th decl. pl.

Table B.1: Nominal morphemes

B.2 Verbal morphemes

The morphemes of the finite verb forms are in table B.2. Apart from these, the morphemes of the nominal forms of the verb are generated using the nominal morphemes: *–nt* with the morphemes of the third declension and *–ns* give the morphemes of the present participle; *–t*, *–tur*, and *–nd* combined with those of the first and second declensions make the morphemes of the perfect participle, future participle, and gerund/gerundive, respectively.

1st. sg.	2nd. sg.	3rd. sg.	1st. pl.	2nd. pl.	3rd. pl.	Description
o	s	t	mus	tis	nt	pres. ind. act.
or	ris	tur	mus	mini	ntur	pres. ind. pass.
bam	bas	bat	bamus	batis	bant	impf. ind. act.
bar	baris	batur	bamur	bamini	bantur	impf. ind. pass.
bo	bis	bit	bimus	bitis	bunt	fut. act. (1st conj.)
bor	beris	bitur	bimur	bimini	buntur	fut. pass. (1st conj.)
em	es	et	emus	etis	ent	fut. act. (3rd, 4th conj.)/pres. subj. act. (1st conj.)
er	eris	etur	emur	emini	entur	fut. pass. (3rd, 4th conj.)/pres. subj. pass. (1st conj.)
am	as	at	amus	atis	ant	pres. subj. act. (2nd–4th conj.)
ar	aris	atur	amur	amini	antur	pres. subj. pass. (2nd–4th conj.)
rem	res	ret	remus	retis	rent	impf. subj. act.
rer	reris	retur	remur	remini	rentur	impf. subj. pass.
i	isti	it	imus	istis	erunt	perf. ind.
erim	eris	erit	erimus	eritis	erint	perf. subj.
eram	eras	erat	eramus	eratis	erant	pqp. ind.
issem	isses	isset	issemus	issetis	issent	pqp. subj.
ero						futex. 1st. sg.

Table B.2: Verbal morphemes

References

Brants, T. (1999). *Tagging and Parsing with Cascaded Markov Models*. Ph. D. thesis, Universität des Saarlandes, Saarbrücken.

Brants, T. (2000). TnT - a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing (ANLP-2000)*, Seattle, WA, pp. 224–231. Association for Computational Linguistics.

Clifford, P. (1990). Markov random fields in statistics. In G. Grimmett and D. Welsh (Eds.), *Disorder in Physical Systems: A Volume in Honour of John M. Hammersley*, pp. 19–32. Oxford University Press.

Diestel, R. (1991). *Graph Theory*. Graduate texts in mathematics. New York: Springer.

Ernout, A. (1953). *Morphologie Historique du Latin* (3rd ed.). Klincksieck.

Gildea, D. (2001). Corpus variation and parser performance. In L. Lee and D. Harman (Eds.), *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pp. 167–202.

Halácsy, P., A. Kornai, and C. Oravecz (2007). HunPos - an open source trigram tagger. In *ACL*, pp. 209–212. Association for Computer Linguistics.

Haug, D. T. T., M. L. Jøhndal, H. M. Eckhoff, E. Welo, M. J. B. Hertzenberg, and A. Müth (2009). Computational and linguistic issues in designing a syntactically annotated parallel corpus of indo-european languages. *Traitement Automatique des Langues 50*(2), 17–45.

Jurafsky, D. and J. H. Martin (2008). *Speech and Language Processing* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Lafferty, J., A. McCallum, and F. Pereira (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 282–289.

Lavergne, T., O. Cappé, and F. Yvon (2010). Practical very large scale CRFs. In *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 504–513. Association for Computational Linguistics.

Pearl, J. (1988). *Probabilitic Reasoning in Intelligent Systems: Networks of Plausible Inference* (2nd (revised) ed.). The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann.

Poudat, C. and D. Longrée (2009). Variation langagières et annotation morphosyntaxique du latin classique. *Traitement Automatique des Langues 50*(2), 129–148.

Samuelsson, C. (1996). Handling sparse data by successive abstraction. In *Proceedings of the 16th conference on Computational linguistics*, Volume 2 of *COLING '96*, pp. 895–900. Association for Computational Linguistics. informal publication.

Schmid, H. and F. Laws (2008). Estimation of conditional probabilities with decision trees and an application to fine-grained pos tagging. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pp. 777–784. Association for Computational Linguistics.

Wallach, H. (2002). Efficient training of conditional random fields. Master's thesis, University of Edinburgh.

Wallach, H. M. (2004). Conditional random fields: An introduction. Technical Report MS-CIS-04-21, University of Pennsylvania.

Walpole, R. E., R. H. Myers, S. L. Myers, and K. Ye (2007). *Probability & statistics for engineers and scientists* (8th ed.). Pearson Education.

Woods, A., P. Fletcher, and A. Hughes (1986). *Statistics in language studies*. Cambridge textbooks in linguistics. Cambridge university press.